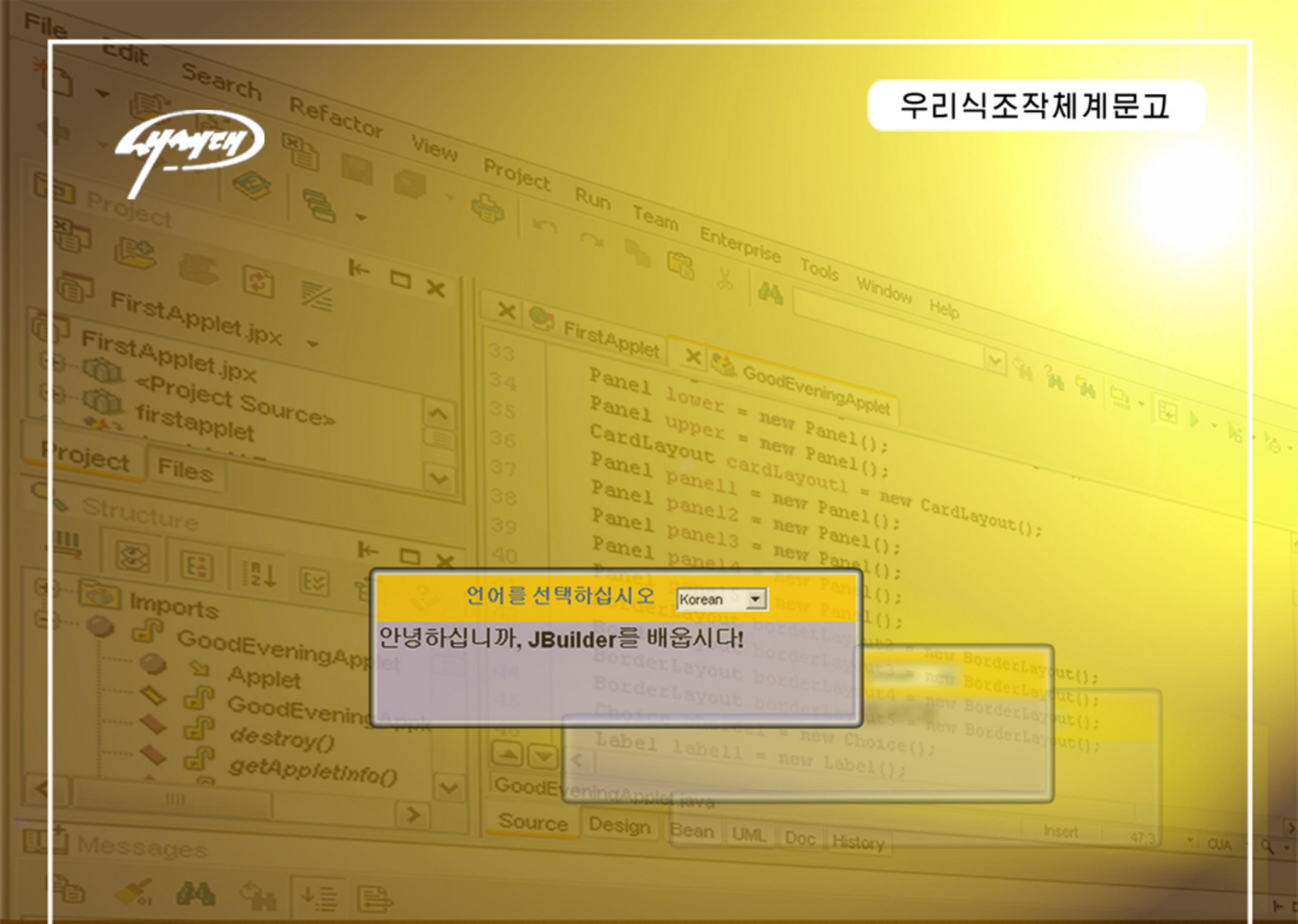
우리식조작체계문고



JBuilder

배우기

교육성교육정보쎈터

주체97(2008)

차 례

머 리 말

위대한 령도자 **김정일**동지께서는 다음과 같이 지적하시였다.

《정보기술, 나노기술, 생물공학은 현시대 과학기술발전의 핵심기초기술입니다.》

21세기는 정보산업의 시대이다. 과학기술발전의 핵심기초기술의 하나인 정보기술을 빨리 발전시켜야 다른 응용기술분야들을 획기적으로 발전시킬수 있고 정보산업시대의 요구에 맞게 경제를 현대화, 정보화할수 있으며 사회경제생활전반을 높은 수준에로 끌어올릴수 있다.

정보기술분야에서 프로그람기술을 빨리 발전시키는것은 장치기술의 발전에 못지 않게 중요하다. 프로그람기술은 투자가 적게 드는 분야이며 경제적효과성이 높으므로 객체화되고 부품화된 규모가 큰 쏘프트웨어들을 대대적으로 개발하는것이 하나의 추세로 되고있다. 프로그람을 질량적으로 빨리 개발하자면 그 개발도구들에 정통하여 그것들을 능숙하게 다루고 창조적으로 응용할수 있어야 한다.

JBuilder는 Java통합개발환경으로서 많은 프로그람개발자들이 사용하고있는 프로그람개발도구이다. 현재 JBuilder는 Java언어에 기초하여 여러 분야의 많은 응용프로그람들을 손쉽게 빨리 개발할수 있는 다양한 도구들을 제공하고있으며 그 판본이 끊임없이 갱신되여 많은 사용자들을 획득함으로써 Java개발도구분야에서 확고한 지위를 차지하고있다.

이 참고서에서는 JBuilder에 대한 간단한 소개와 쏘프트웨어의 설치방법, 개별적인 부품들의 기능과 사용방법, JBuilder에 의한 프로그람개발수법들에 대하여 소개하였다. 특히 다중스레드기술, 망통신기술, JSP기술 등 여러 분야의 프로그람개발방법론들을 구체적으로 설명하고있다. 또한 독자들의 리해를 돕기 위하여 매 구체적인 개발단계마다 그림을 삽입하면서 설명을 하였으며 필요한 프로그람원천코드들을 소개하였다.

JBuilder와 관련하여 깊이 주어야 할 많은 내용들과 개발수법들이 있지만 이 책에서는 일반독자들을 대상으로 프로그람작성방법론을 기본적으로 주는것을 목적으로 하여 내용을 구성하였다. 독자들의 좋은 의견이 있기를 바란다.

제1장. JBuilder의 개요

Borland회사는 최근 Java ALM(Application Life Management)해결방안을 내놓았다. 이것은 Java개발을 보다 편리하게 하고 Java응용프로그람개발을 사용자일반으로부터 기업화된 개발에로 전환시켰다. 다른 쏘프트웨어기반에 비하여 J2EE에 기초한 Borland회사의 쏘프트웨어기반은 기술측면에서 확고히 우세를 차지하고있으며 응용프로그람의 개발속도를 높이고있다. JBuilder는 주류를 이루고있는 응용프로그람봉사기들과 밀접히 통합되여있다. 여기에는 Borland Enterprise Server, BEA WebLogic, IBM WebSphere, Oraclexi Application Server, Sybase EAServer, Sun ONE Application Server 등이 포함된다. JBuilder는 또한 주콤퓨터나 원격콤퓨터에서 Enterprise JavaBean을 실행하고 오유제거할수 있으며 Borland회사의 다른 쏘프트웨어제품들과 유기적으로 결합되여있다.

Borland JBuilder는 현재 쏘프트웨어시장에서 일정한 지위를 가지고있는 교차기반개발환경으로서 주로 업무능력을 갖춘 기업급의 Java응용프로그람들을 만드는데 리용되고있다. JBuilder는 또한 표준적인 Java최신판본을 지원하며 개발주기를 단축하고있다. 이 책에서는 JBuilder2005에 대하여 취급하였다. JBuilder의 시각적인 통합개발환경에는 본문편집기, 프로젝트작성도구, 구조판과 오유제거기 등이 포함되여있다. 이 장에서는 JBuilder의 몇가지 기능들과 설치방법 및 개발환경에 대하여 서술한다.

제1절. JBuilder의 기능

JBuilder는 주로 다음과 같은 기능들을 갖추고있다.

- J2EE기반응용프로그람을 개발하고 배비(deploy)한다.

교차기반개발환경으로서 설계, 개발, 오유제거, 검사로부터 배비에 이르기까지 모든것을 할수 있다.

- 웨브봉사(Web Service)개발을 가속화한다.

최신의 웨브봉사기술을 지원하며 SOAP, WSDL, UDDI, WSIL들을 포함하고있다. 웨브봉사의 지원과 관련한 SOAP봉사기조수, TCP관찰기, WSDL입력조수, EJB조수, Web Service Explorer를 가지고있으며 많은 웨브봉사실례들을 제공하고있다.

- 복잡한 기업급의 개발환경들을 결합시킨다.

Apache Ant 1.5 등을 제공하여 개발환경을 지원한다.

- 이미 있는 프로젝트들을 리용할수도 있으며 UML도해를 사용하여 프로그람코드를 시각적으로 전개할수도 있다.

프로그람코드로부터 UML모형을 빨리 생성할수 있고 UML클라스도해를 통하여 프로그람코드를 시각화할수 있다.

- 기업급의 단위검사기능과 강력한 오유제거기를 사용하여 품질이 높은 응용프로그람을 개발할수 있다.

JUnit와 밀접히 결합되여있으며 프로그람의 작성과 실행, 프로그람단위별 검사분석은 물론 프로젝트의 조직까지도 할수 있다. 또한 내부적으로 구축된 JNDI, JDBC와 대비검사를 진행하여 프로그람작성시간을 단축하고있다. HotSwap를 사용하여 프로그람을 수정할수 있다. 또한 JSR-45을 지원하며 현지에서 또는 원격으로 Java언어가 아닌 다른 언어로 된 프로그람코드의 오유제거를 방조할수 있다.

- 내장된 개발팀환경을 리용하여 개발팀의 생산능률을 높일수 있다.

개발팀사이의 련계를 강화하고 생산능률을 높이기 위하여 JBuilder는 팀개발환경을 제공하고있다. 즉 Borland StarTeam, Rational ClearCase, CVS와 Microsoft Visual SourceSafe 등의 판본조종과 쏘프트웨어 배비관리체계를 지원한다. 여기서 StarTeam은 아주 중요하고도 협동합작할수 있는 지식기지를 제공하고있다. 이것은 팀성원이 서로 다른 도구와 흐름공정에서 오는 정보를 공유하여주며 각종 항목의 생명주기과제를 지원한다.

- JBuilder는 표준적인 Java최신판본과 잘 알려진 쏘프트웨어업체의 쏘프트웨어들을 지원하고있어 Java개발환경을 다른 새로운 경계분야로 이끌어가고있다.

JBuilder는 표준적인 Java최신판본을 지원한다. 여기에 Java2, Java 2Swing/JFC, XML, Java2D, Java Collections, Message Queue, Accessibility API, JavaBean, JDBC, Enterprise JavaBean, JSP/Servlet, Serialization, Inner Class, RMI, JNI, Java Archives 등이 포함된다. OpenTools API를 사용하여 JBuilder를 전용화하고 확장할수 있으며 또한 JBuilder사가 개발한 OpenTools쏘프트웨어의 끼워넣기(Plug-In)를 사용할수도 있다.

- Apache Structs, Servlets, JSP, XML을 사용하여 동적인 웨브응용프로그람들을 만들고 오유제거할수 있다.

제2절. JBuilder의 설치

JBuilder의 설치에 대하여 소개하기 전에 우선 JBuilder의 제품들과 하드웨어에 대한 요구에 대하여 고찰한다.

1.2.1. JBuilder의 제품들

JBuilder에는 크게 4개의 제품이 있다. 즉

- JBuilder Enterprise(기업판)

- JBuilder Developer(개발자판)
- JBuilder Personal(개인판)
- JBuilder WebLogic Edition(WebLogic기업판)

1) 기업판

기업판은 Optimizeit Suite와 밀접히 통합되여있으며 사용자가 속도가 최량이고 믿음성있으며 확장성이 큰 응용프로그람의 개발을 할수 있도록 한다. 기업판은 설계, 프로그람작성, 배비 등을 포괄하고있으며 Web Service, EJB, Web, XML, 자료기지응용과 관련된 기업급 Java응용프로그람들을 포함하고있다.

Borland Enterprise Studio for Java는 완전한 생명주기해결방안으로서 모형작성으로부터 Java응용프로그람과 전자업무의 실현에 이르기까지 모든 개발과정을 포괄하고있다.

2) 개발자판

개발자판은 전문프로그람작성자지향의 제품으로서 시간단축도구들 례하면 오유제거기능, JDK절환, Javadoc도구와 팀개발관리도구 등을 포함하고있다.

3) 개인판

개인판은 Java언어를 학습하여 프로그람을 개발하려는 학생이나 상업적용도를 목적으로 하지 않는 일반사용자를 위한 소규모의 제품이다.

4) WebLogic기업판

JBuilder는 EJB, Web, XML, 자료기지응용 등을 제공하며 J2EE기반에서 BEA WebLogic에 빨리 배비하게 한다.

JBuilder는 시각적인 UML코드를 지원할뿐아니라 단위검사를 재생할수도 있으며 통일적이고 표준적이며 간단하고 신축성있는 골격구조를 제공하여 프로그람개발효률을 높인다. BEA WebLogic기반은 기업급의 IT조직이 제품생산력을 높이고 원가를 낮추게 한다.

1.2.2 JBuilder의 하드웨어에 대한 요구

제품마다 체계의 쏘프트-하드웨어요구가 서로 다르며 이러한 불일치는 주로 기억기와 하드디스크공간에 대한 요구측면에서 나타난다. 아래에서 요구사항을 고찰한다.

1) Windows에서의 요구사항

- CPU: Intel펜티움Ⅱ/233MHz이상(PⅡ400MHz이상을 사용할것을 요구한다.)
- 조작체계: Microsoft Windoews 2000(SP2), Windows XP, Windows NT 4.0(SP6a) 또는 그 이상 판본의 조작체계

2) Linux에서의 요구사항

- CPU: Intel펜티움Ⅱ/233MHz이상(PⅡ400MHz이상을 사용할것을 요구한다.)
- 조작체계: 기정인 GNOME, KDE창문의 Red Hat Linux 7.2, Red Hat Enterprise

Linux WS 2.1 또는 그 이상의 조작체계

3) Solaris에서의 요구사항

- CPU: UltraSPARC Ⅱ이상/G3 processor 350MHz이상
- 조작체계: Solaris 8(2.8) 또는 그 이상의 조작체계

4) CD-ROM구동기

5) 마우스와 기타 입력장치

6) 분해능 1024×768pixel과 256색이상의 VGA영상표시장치

7) 최소 700MB의 하드디스크공간(설치시 요구되는 공간을 포함)

1.2.3. 설치과정

Windows와 Linux조작체계에서의 설치과정이 거의 류사하므로 Windows조작체계에서의 설치과정만을 고찰한다. 여기서는 또한 JBuilder Enterprise(기업판)를 가지고 JBuilder의 설치과정을 설명한다. JBuilder의 설치프로그람은 자동적으로 하드웨어장치를 검사하며 최량인 설정상태로 조절할수 있다. 구체적인 설치단계는 다음과 같다.

단계

1 JBuilder설치용CD를 구동기에 넣는다. 뿌리등록부의 《install_windows.exe》파일을 두번 찰칵한다. Linux조작체계인 경우는 install_linux.sh을 실행한다. 그러면 그림 1-1과 같은 설치대면부가 자동적으로 나타나며 이 대면부에서 설치하려는 Borland회사의 제품을 선택할수 있다. 다음 【Install Borland JBuilder 2005 Enterprise】단추를 찰칵한다.



그림 1-1. JBuilder설치시작대화칸

2 이때 Borland JBuilder Enterprise설치조수대화칸이 나타난다.(그림 1-2) 자기가 요구하는 부품을 선택한 다음 【Install】단추를 찰칵한다.

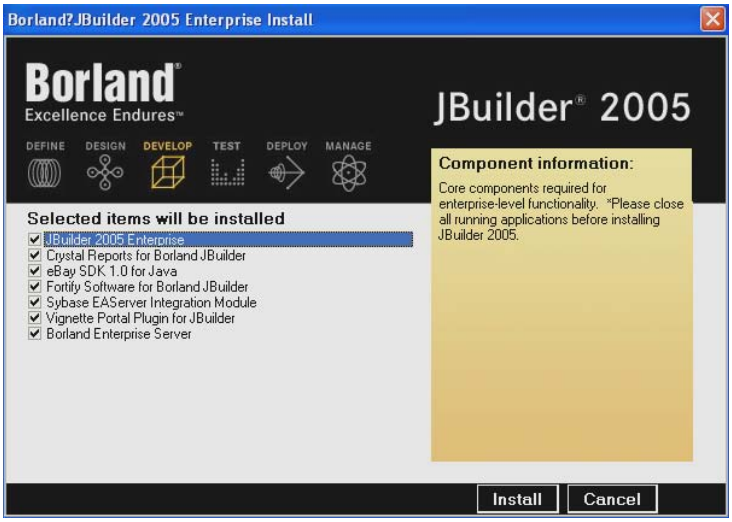


그림 1-2. 설치항목을 선택하는 대화칸

3 이때 그림 1-3과 같은 설치준비대화칸에 들어가며 【Next】단추를 찰칵한다.

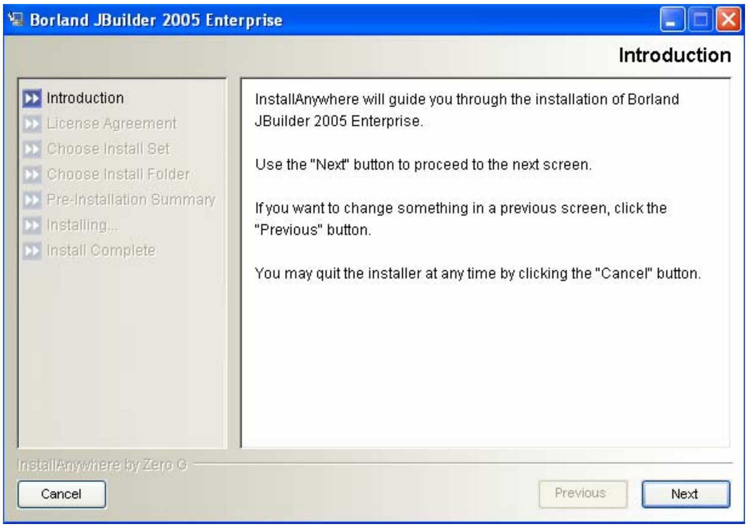


그림 1-3. 설치준비대화칸

4 그림 1-4와 같은 사용허가합의대화칸에 들어가 허가합의내용을 읽어보아야 한다. 《I accept the terms of the License Agreement》항목을 선택한 다음 【Next】단추를 찰칵한다.



그림 1-4. 설치를 위한 사용허가합의대화칸

5 그림 1-5와 같이 Borland JBuilder Enterprise의 설치방식에서 《Full Install》를 선택한 다음 【Next】단추를 찰칵한다.



그림 1-5. JBuilder2005 Enterprise의 설치방식선택

6 그림 1-6과 같이 Borland JBuilder Enterprise의 설치경로를 선택한다. 이때 기정경로를 사용할수도 있고 【Choose...】단추를 찰칵하여 자기가 설치하려는 경로를 선정할수도 있다. 다음 【Next】단추를 찰칵한다.



공백문자가 있는 파일등록부를 사용하지 말아야 한다. 그것은 JBuilder가 이러한 등록부에서는 실행할수 없기때문이다.

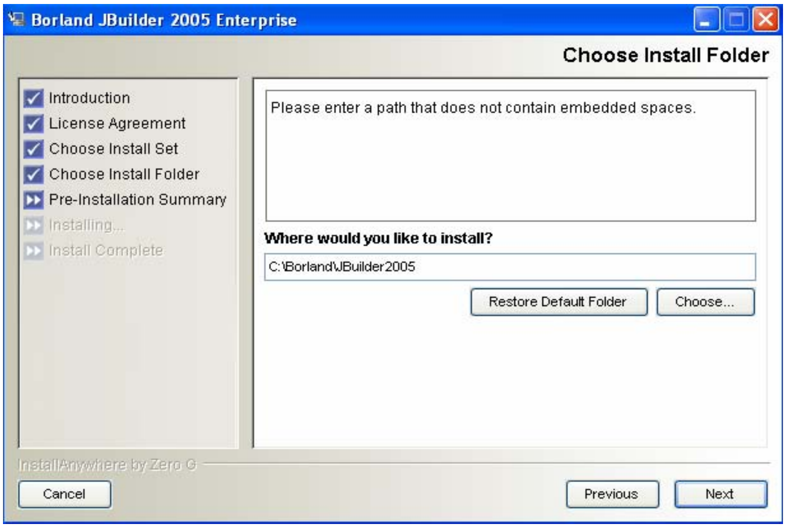


그림 1-6. JBuilder2005 Enterprise의 설치경로선택

7 이때 Borland JBuilder Enterprise의 예비설치대화칸에 들어간다. 이 대화칸에서 마지막으로 선택한 내용이 정확한가를 확인하고 【Install】단추를 찰칵한다.



그림 1-7. JBuilder2005 Enterprise의 예비설치대화칸

8 설치과정을 그림 1-8에서 보여준다. 필요에 따라 설치과정에 【Cancel】단추를 찰

칵하여 Borland JBuilder Enterprise의 설치를 중단하고 취소할수 있다.



그림 1-8. JBuilder2005 Enterprise의 설치과정

9 설치완료후에는 【Done】단추를 찰칵하여 설치를 완전히 끝마친다.

제3절. JBuilder의 통합개발환경

JBuilder의 통합개발환경이란 무엇인가? 아래에서는 AppBrowser를 구체적으로 소개한다.(그림 1-9)

그림 1-9는 JBuilder를 실행시킨 후에 처음으로 보게 되는 창문이다. 여기서 보는바와 같이 JBuilder는 대면부를 통해 편집, 파일 및 프로젝트관리, 시각적인 대면부설계, 열람, 콤파일, 오유제거와 기타 조작을 수행할수 있는 포괄적인 개발기반이다. 보통 이것을 응용프로그람열람기(AppBrowser)라고 부른다. 매 부분의 이름과 그 기능에 대하여 아래에서 설명한다.

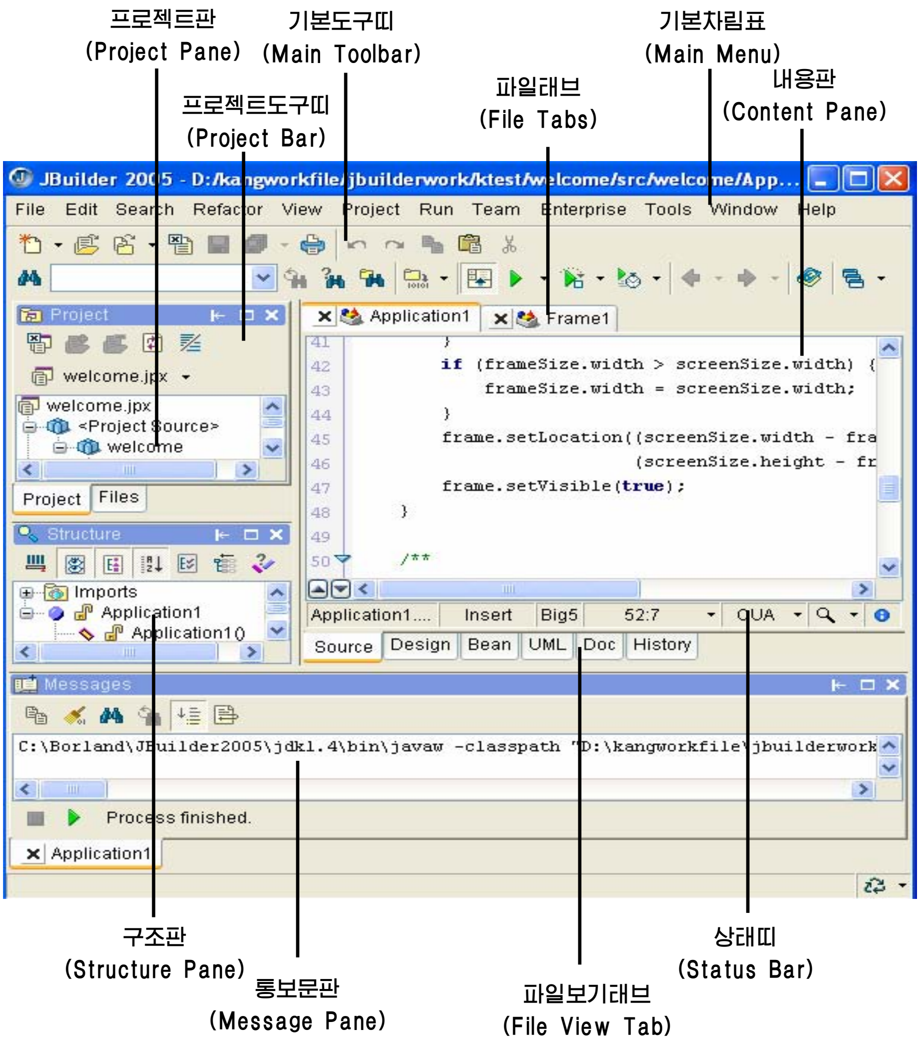


그림 1-9. JBuilder2005의 통합개발환경

- 기본차림표(Main Menu)

기본차림표는 프로그람작성에서 필요한 작업들을 할수 있게 한다. 례하면 프로젝트와 파일의 열기와 보관, 파일에서의 본문조사, 콤파일, 오유제거 등을 할수 있다.

- 기본도구띠(Main Toolbar)

기본도구띠는 기능에 따라 몇개의 작은 도구띠들로 구분한다. 매개 아이콘들은 차림표지령에 대한 지름접근을 제공하고있다.

- 프로젝트판(Project Pane)

프로젝트판은 선택한 프로젝트들의 내용들을 현시하며 프로젝트들을 열람하고 조작하는데 쓰인다.

- 프로젝트도구띠(Project Bar)

프로젝트도구띠는 여러가지 아이콘을 포함하고있는데 그것은 파일을 추가, 삭제하거나 작업하려고하는 프로젝트파일의 열기, 닫기, 갱신하는데 쓰인다.

- 구조판(Structure Pane)

구조판은 현재 파일의 구조를 현시한다. Java파일에 대해서 본다면 이것은 계층구조로 모든 메쏘드, 속성, 사건들을 보여주고있다. 구조판은 아래방향으로 전개할수 있으며 여기에 있는 클라스나 대면을 두번 찰각하면 내용판에서 그의 구체적인 코드를 볼수 있다.

- 내용판(Content Pane)

내용판에는 열려진 파일들의 내용이 현시된다. 매 파일은 파일이름을 나타내는 태브와 그 아래부분에 있는 여러가지 보기태브들을 가진다.

- 파일보기태브(File View Tabs)

내용판에 현시된 파일의 각종 보기태브 례하면 Source, Design, Bean, UML, Doc, History보기태브들이 있다.

- 파일태브(File Tabs)

파일태브는 열려진 파일이름을 현시하는데 쓰인다. JBuilder는 프로젝트에 관한 파일태브만을 현시한다. 열려진 파일중에서 구체적인 내용을 보려고 하는 파일의 태브를 찰각하면 그 파일이 내용판에 현시된다.

주의

파일이 수정되였을 때 파일태브상에 단추가 생겨나며 수정한것이 없을 때에는 단추가 생겨난다.

- 통보문판(Message Pane)

통보문판은 태브페지들에서 있게 되는 통보문들을 현시하는 구역이다. 례하면 설계기, 탐색결과, 콤파일기, 오유제거기 및 실행스레드로부터 생겨나는 통보문들을 현시한다.

- 상태띠(Status Bar)

상태띠에는 스레드관련 및 그 결과에 대한 최신정보가 현시된다.

- Design방식과 부품선택판(Component Palette)

만일 UI설계기에서 파일을 보려면 내용판의 Design태브페지를 보면 된다. Design페지에서 사용자대면부를 작성할수 있으며 부품선택판은 오직 Design페지에서만 사용할수 있다.(그림 1-17)

사용자대면을 작성하려면 부품선택판의 부품을 내용판에 배치하든가 또는 구조판을 리용할수도 있다. 그러면 JBuilder는 자동적으로 생성한 코드를 파일에 삽입하게 된다. Inspector창문을 사용하면 부품의 련관속성을 조정할수 있다.

제4절. 간단한 프로젝트의 작성

이 절에서는 한가지 실례를 가지고 JBuilder환경에서의 Java응용프로그람개발에 대한 기본과정을 소유하도록 한다.

1.4.1. Project Wizard를 리용한 프로젝트의 작성

JBuilder의 Project Wizard를 사용하여 프로젝트를 작성할수 있다. firstapp.jpx인 프로젝트를 작성해보기로 하자. 구체적인 작성단계는 다음과 같다.

단계

1 【File】차림표의 【New Project】지령을 선택하면 그림 1-10과 같은 대화칸이 나타난다.

2 【Name:】본문칸에서 《untitled1》을 《firstapp》로 바꾸고 다른 추가선택항목들은 기정으로 설정한다.

3 【Next】단추를 찰칵하여 다음 페지에 들어간다. 이때 그림 1-11과 같은 대화칸이 나타난다.



그림 1-10. Project Wizard대화칸의 첫 페지

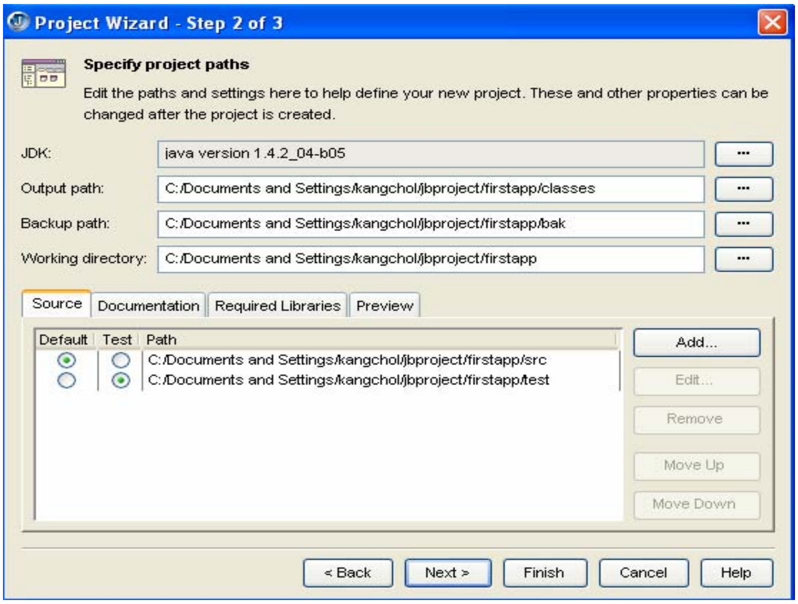


그림 1-11. Project Wizard대화칸의 2번째 페지

4 이 페지에서는 추가선택항목들을 기정으로 한다. 【Next】단추를 찰칵하면 그림 1-12와 같은 대화칸이 나타난다. 【Title:】마당에 《firstapp》을 입력하고 【Finish】단추를 찰칵하여 프로젝트작성을 끝낸다. 이때 JBuilder는 firstapp프로젝트를 자동생성한다.

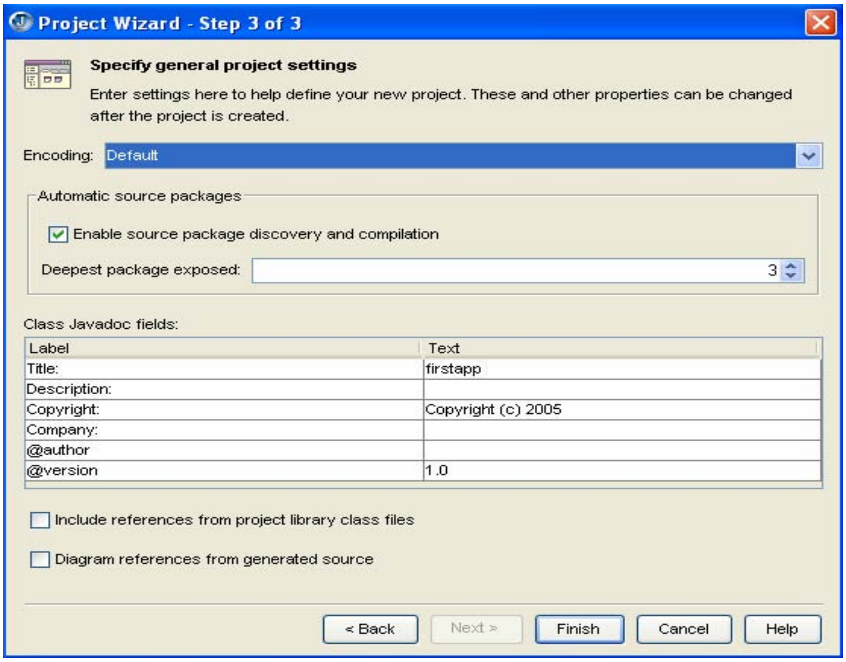


그림 1-12. Project Wizard대화칸의 3번째 페지

1.4.2. Application Wizard를 리용한 Java응용프로그람의 작성

여기서는 Application Wizard를 사용하여 Java응용프로그람을 작성한다. 구체적인 단계는 아래와 같다.

단계

1 【File】차림표의 【New】지령을 선택하면 그림 1-13과 같은 대화칸이 나타난다. 이 대화칸에서 서로 다른 파일들을 작성하는데 쓰이는 여러가지 조수들을 볼수 있다.



그림 1-13. New추가선택항목대화칸

2 Application아이콘을 두번 찰칵하면 Application Wizard대화칸이 나타난다.(그림 1-14) Application Wizard대화칸에 기정적으로 설정되여있는 추가선택항목들이 현시된다. 【Next】단추를 찰칵하여 2번째 페지로 들어간다.(그림 1-15)



그림 1-14. Application Wizard대화칸의 첫 페지

3 이 대화칸에서 【Class:】본문칸에 《firstappFrame》을 입력하고 【Title:】본문칸에 《This is my Java application》을 입력한다.

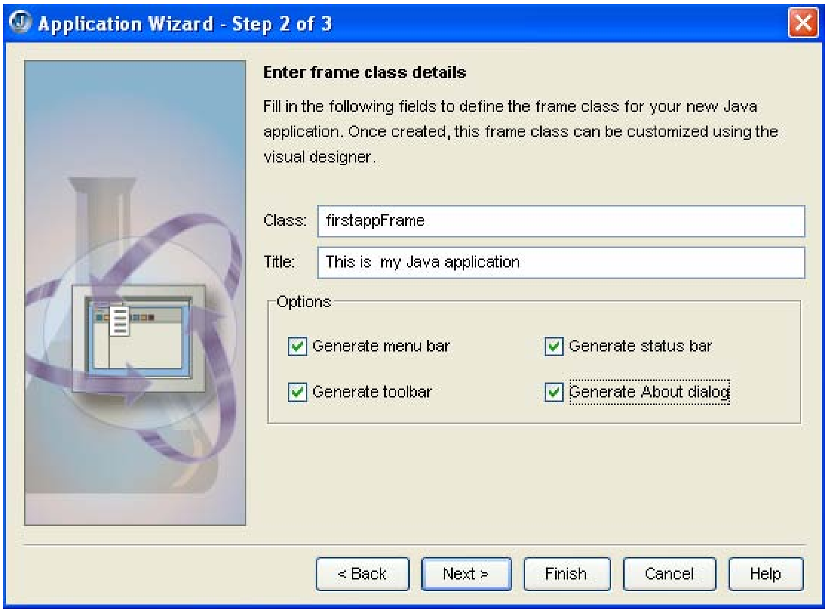


그림 1-15. Application Wizard대화칸의 2번째 페지

4 【Generate menu bar】검사칸을 찰칵하면 기정인 차림표띠를 자동생성한다.

5 【Generate toolbar】검사칸을 찰칵하면 기정인 도구띠를 자동생성한다.

6 【Generate status bar】검사칸을 찰칵하면 기정인 상태띠를 자동생성한다.

7 【Generate About dialog】검사칸을 찰칵하면 기정인 정보소개대화칸을 자동생성한다.

8 【Next】단추를 찰칵하여 그림 1-16과 같은 페지로 들어간다.



그림 1-16. Application Wizard대화칸의 3번째 페지

9 Application Wizard대화칸의 3번째 페지에서 추가선택항목들을 기정으로 선택하고【Finish】단추를 찰칵하면 Java응용프로그람의 설치가 끝난다. 이때 Application1.java, firstappFrame.java와 firstFrame_AboutBox.java인 3개 프로그람이 자동적으로 만들어져 프로젝트에 추가된다. 이 프로그람들은 응용프로그람대면부와 그에 대응하는 원천프로그람을 자동생성하여 얻은것이다.

자동생성과정을 끝마친 후 Design태브를 찰칵하면 그림 1-17과 같은 화면이 얻어진다. Inspector에는 현재 프로그람의 속성과 사건들을 현시하는 속성(property)창문과 사건(event)창문이 있다.

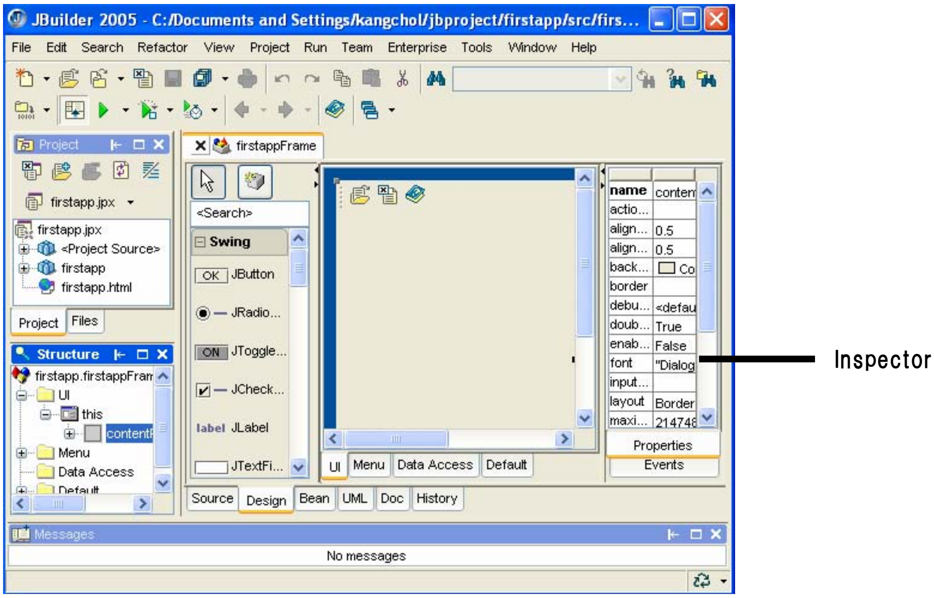


그림 1-17. 자동생성된 응용프로그람대면부

1.4.3. 차림표의 제작

그림 1-17에서 체계가 자동생성한 프레임과 차림표를 보았다. 그러나 이것으로 응용프로그람이 완성되는것이 아니다. 자기에게 필요한 차림표들을 더 추가하여야 한다. JBuilder에서 차림표의 제작은 아주 간단하다. 우선 구조판에서 Menu부품아이콘을 두번 찰칵하여 차림표설계기를 열면 그림 1-18과 같은 대면부가 나타난다.

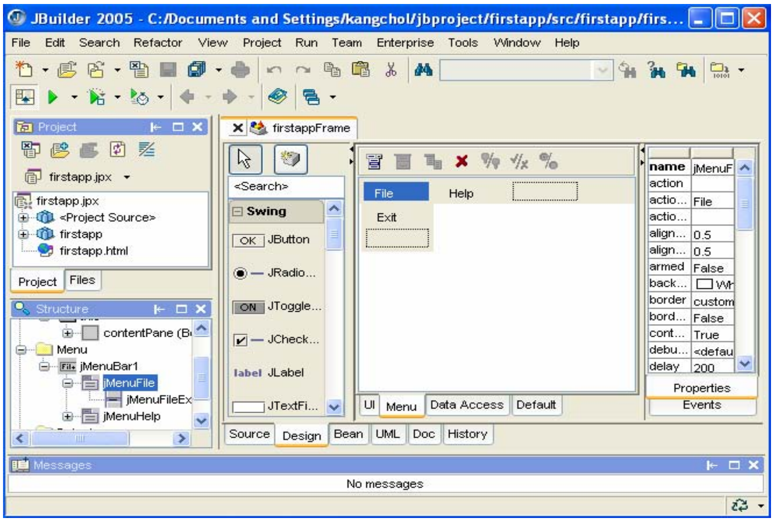


그림 1-18. 차림표의 설계

차림표설계기에서는 차림표작성을 쉽게 할수 있다.

Java에서 차림표는 여러 형식을 가질수 있다.
또한 아이콘을 추가할수도 있다. 추가방법은 다음과 같다. 우선 프로젝트에 화상파일을 추가하고 속성창문의 icon속성을 찰칵한다. 내리펼침칸을 찰칵하면 방금 추가한 화상파일을 볼수 있다.

이 실례에서는 한개의 차림표만을 추가한다. 구체적인 단계는 다음과 같다.

단계

1 【Help】차림표를 선택하고 마우스오른쪽단추를 찰칵하여 【Insert Menu】지령을 선택한다. 그러면 【Help】차림표의 옆에 새로운 차림표가 만들어진다. 다음 새로 만들어진 차림표를 두번 찰칵하여 차림표이름으로 《new》를 입력한다. 다시 이 차림표의 빈 부분차림표를 두번 찰칵하여 《Click Me》라고 부분차림표의 이름을 입력한다.

2 차림표에 아이콘을 추가하기 위하여 우선 하나의 gif화상파일을 프로젝트에 추가해야 한다. 방법은 【Project】차림표의 【Add】/【Add Files/Packages/Classes...】지령을 선택하면 된다.(그림 1-19)

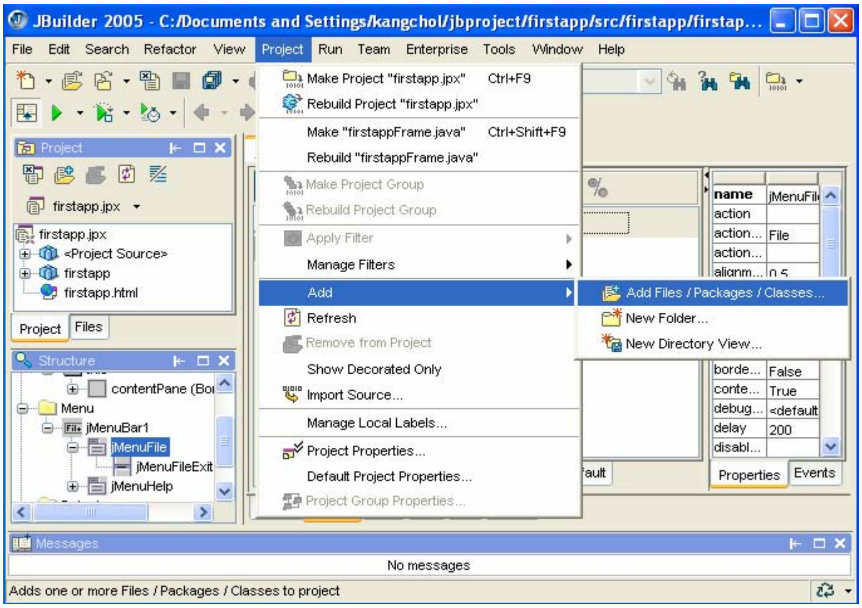


그림 1-19. 프로젝트에 파일을 추가

3 이때 그림 1-20과 같은 《Add to "firstapp.jpx"》대화칸이 나타난다. 추가하려는 화상파일을 찾아 선택한 다음 【Ok】단추를 찰칵하면 프로젝트에 이 화상이 추가된다.

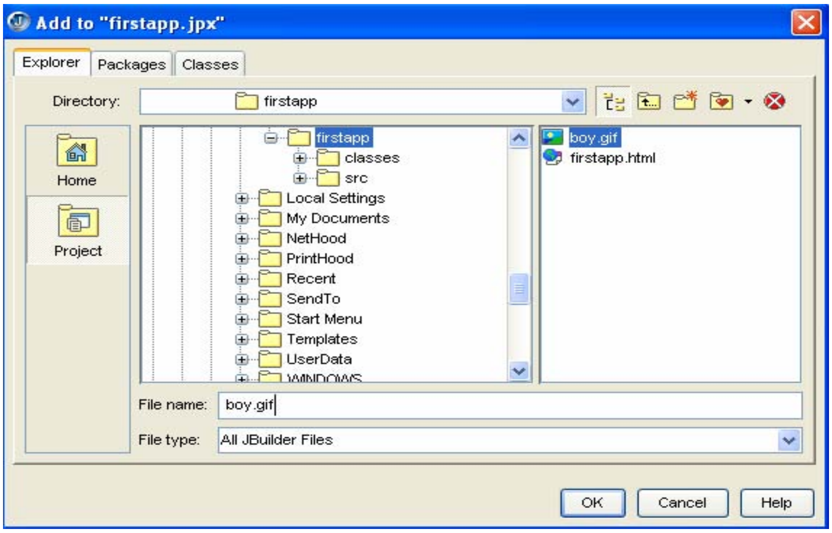


그림 1-20. Add to “firstapp.jpx” 대화칸

4 【New】차림표의 【Click Me】부분차림표를 선택하고 icon속성의 아래방향화살표를 찰칵하여 방금 추가한 화상이름을 선택한다. 여기서 화상파일이름은 《 boy.gif》이다.(그림 1-21)

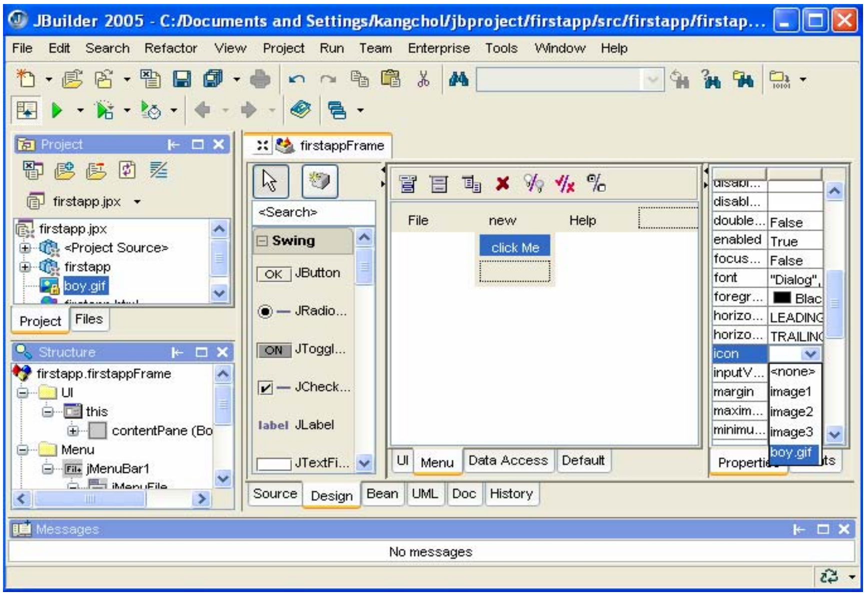


그림 1-21. Icon객체의 추가

5 이때 체계는 《 jMenuItem1.setIcon(new ImageIcon(new java.net.URL (file:///C:/JBuilder2005/1/firstapp/boy.gif)));》라는 코드를 자동적으로 추가한다.

6 차림표작성이 끝난 다음 【Run Project】지령을 선택하거나 지름건 【F9】을 사용하여 결과를 얻는다.(그림 1-22)

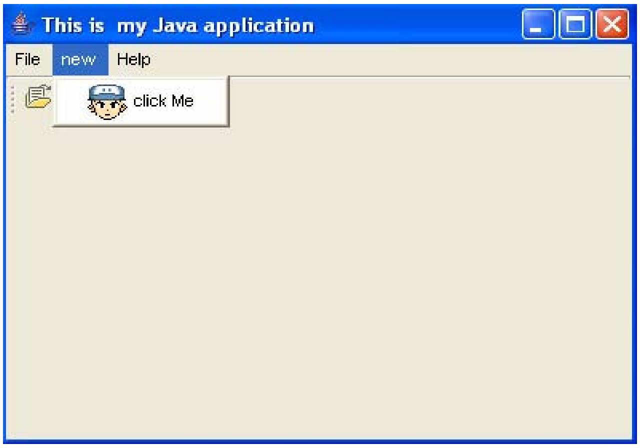


그림 1-22. 차림표

1.4.4. 대면부의 설계

대면부의 설계과정에 대하여 보기로 하자.

- 총적계획

대규모프로젝트에 대한 쏘프트웨어의 설계에서는 우선 리용한 프레임과 그이름, 작용 등의 속성을 확정하여야 한다. 파일, 프레임, 부품에 관하여 통일적인 이름짓기규칙을 정해야 한다. 이렇게 하면 프로그람의 개발뿐아니라 그 유지보수에도 유리하다.

- 대면부의 격식

같은 프로젝트에 대하여 모든 프레임과 부품의 격식은 일치하여야 한다. 이렇게 하여야 사용자가 프로그람대면부를 보다 쉽게 받아들일수 있다. 실례로 Microsoft Office와 Borland JBuilder는 사용자의 기호에 맞게 대면부의 격식이 설계되여있다.

Java는 여러 조작체계기반에서 코드를 실행할수 있으므로 대면부격식 역시 여러가지 형식을 가지고있다. JBuilder에서는 프로그람을 작성할 때 4개의 서로 다른 류형의 격식을 사용할수 있다. 4개의 대면부격식으로서는 Borland, Windows, CDE\Motif와 Metal이 있다. 물론 프로젝트대면부는 한개 류형의 격식만을 사용한다.

대면부격식을 원하는 격식으로 설정하자면 다음과 같이 한다. 즉 UI설계방식으로 절환하고 Design태브를 찰칵한다. 여기서 마우스오른쪽단추를 찰칵하여 지름차림표를 연다. Look and Feel/Metal항목을 선택하면 체계대면부의 격식은 Metal격식으로 변한다.(그림 1-23)

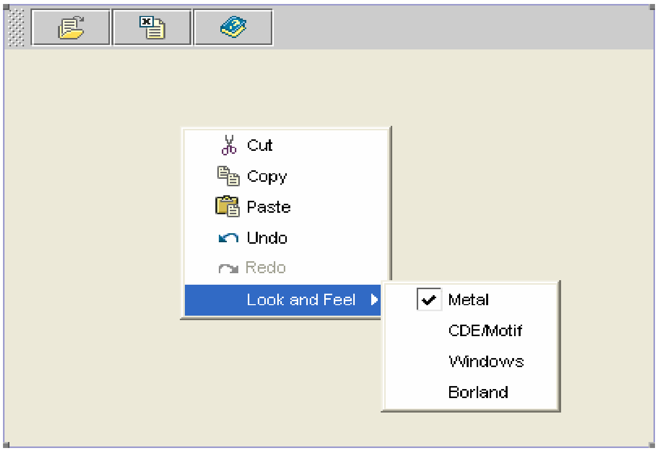


그림 1-23. 체계대면부격식의 설치

그림 1-24와 같이 프레임우에 부품들을 설치해보자. 구체적인 단계는 아래와 같다.

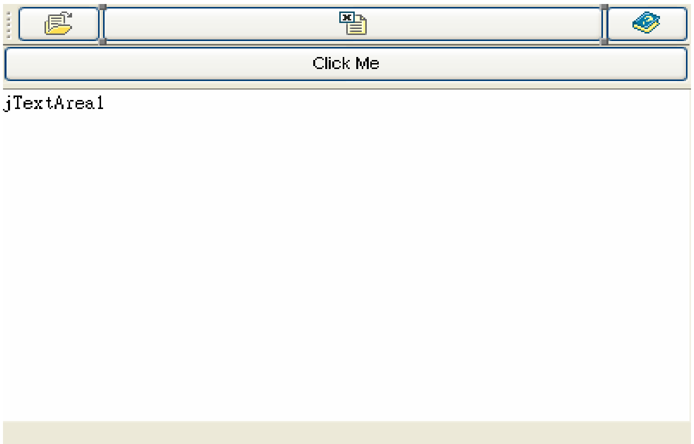


그림 1-24. 부품들의 배치

단계

1 부품선택판의 Swing Containers서고에서 JPanel부품을 선택한다.

2 부품선택판의 옆에 있는 대면부우에 마우스끌기하여 javax.Swing.JPanel1부품객체를 추가한 다음 constraints속성의 내리펼침화살표를 찰칵하여 《Center》를 선택한다.

3 layout속성의 내리펼침화살표를 찰칵하여 《Border layout》를 선택한다.

4 부품선택판에서 Swing Containers서고의 javax.swing.JSplitPanel부품을 선택한다.

5 JPanel1에 JSplitPanel1부품객체를 추가하고 크기와 위치를 조절한다.

6 orientation속성의 내리펼침화살표를 찰칵하여 《VERTICAL_SPLIT》를 선택한다.
7 constraints속성의 내리펼침화살표를 찰칵하여 《Center》를 선택한다.
8 부품선택판에서 Swing서고의 JButton부품을 선택한다.
9 jSplitPanel1부품객체우에 jButton부품객체를 추가한다.
10 text속성을 《Click Me》로 설정한다.
11 부품선택판에서 javax.swing.JTextArea부품을 선택한다.
12 jSplitPanel1부품객체의 아래에 jTextArea부품객체를 추가한다.
13 text속성은 기정으로 설정한다.

이것으로 차림표와 기본대면부의 설계가 완성되였다. 아래에서는 실제적인 프로그람작성을 진행한다. 다시말하여 매 부품에 대한 응답코드를 추가한다.

콤파일, 련결 및 실행은 차림표를 리용하여 진행할수 있다. 【Project】차림표의 【Make Project "firstapp.jpx"】지령을 선택하면 콤파일, 항목련결, 대응하는 집행파일을 생성하며 【Run】차림표의 【Run Project】지령을 선택하면 이 파일에 대한 콤파일 및 련결을 진행하고 생성한 집행파일을 실행한다. 결과는 그림 1-25와 같다.

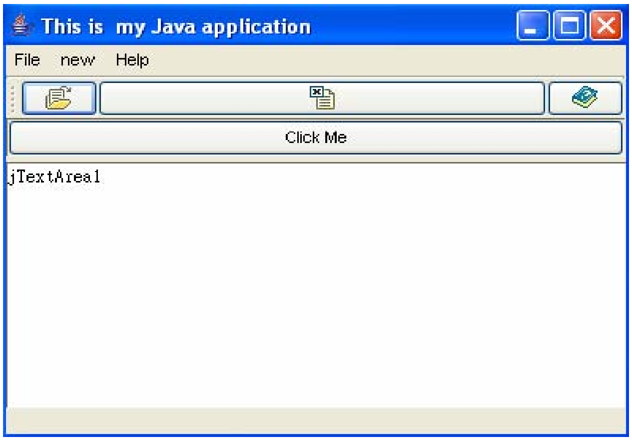


그림 1-25. 기본대면부의 실행결과

1.4.5. 코드의 추가

부품의 응답에 대응하는 프로그람코드를 추가해보자. 아래에서 간단한 실례를 가지고 프로그람코드의 추가과정을 설명한다.

우선 【Click Me】단추를 찰칵할 때 일어나는 사건에 대한 응답을 추가한다. 방법은 다음과 같다.

Designer태브를 찰칵하고 UI태브를 찰칵하면 나타나는 내용판(UI내용판) 또는 구조판의 부품계층구조(Component Tree)에서 【Click Me】단추를 찰칵한다. 계속하여 Inspector에서 사건(event)태브를 찰칵한 다음 actionPerformed사건에서 마우스를 찰

칵하여 이 사건을 능동으로 한다. 이때 코드에 나타나는 기정값은 jButton4_actionPerformed이며 결과는 그림 1-26과 같다. Enter건을 누르면 UIDesigner대면부에 대한 내용이 Source페지에 반영된다.

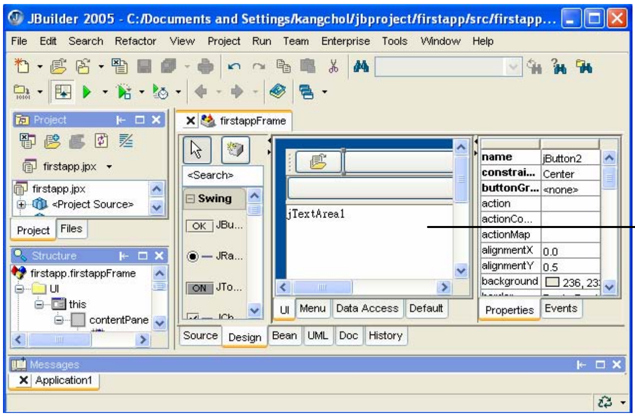


그림 1-26. 코드추가에 대한 현시

이때 jButton4_actionPerformed메쏘드가 창조된다.
그러면 코드편집창문에 다음과 같은 코드가 나타난다.

```
void jButton4_actionPerformed(ActionEvent e){
}
```

《{》와 《}》사이에 《jTextArea1.append("How do you do!");》을 입력하여 JButton4사건에 대한 응답코드를 추가한다. 이것을 실행하면 본문대화칸에 《How do you do!》를 현시한다.

코드의 추가과정을 그림 1-27에서 보여준다.

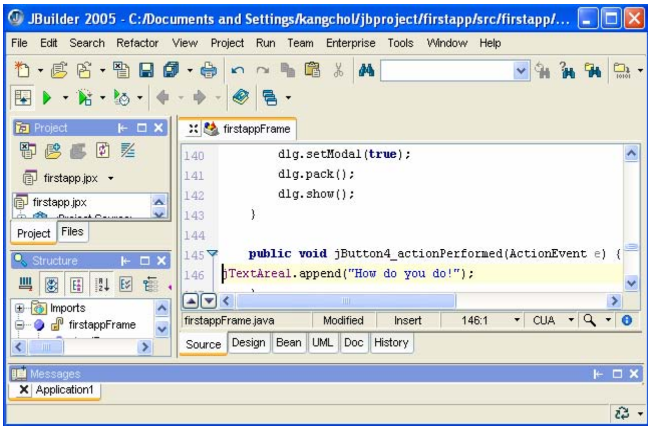


그림 1-27. 코드작성

JBuilder는 코드복용기능도 제공하고있다. 이 기능은 프로그람을 작성하는데서 편리하다.

우선 UI내용판 또는 구조판의 부품계층구조에서 JButton4【Click Me】를 선택한다. 계속하여 사건창문의 actionPerformed사건에서 마우스를 찰칵하여 이 사건을 능동으로 한다. 코드에 나타나는 기정값은 jMenuItem1_actionPerformed이다. 다음에 그것을 그림 1-27과 같이 jButton4_actionPerformed로 고치고 되돌이건을 누르면 UIDesigner대면부의 Source페지에 들어간다. 이때 jButton4_actionPerformed메쏘드는 능동으로 된다. 이렇게 하면 차림표의 【Click Me】부분차림표와 창문안의 【Click Me】단추에 대한 응답은 동일한 결과를 나타내게 된다.

마지막으로 응용프로그람의 실제기능을 추가하여야 한다.

JBuilder가 풍부한 부품을 제공하고 있을지라도 그것은 우리가 해결하려는 모든 문제들을 해결할수 없으며 응용프로그람체계 역시 다양하므로 몇개의 부품들의 조합으로 작성될수 없다. 그러나 JBuilder는 풍부한 부품들을 가지고있으므로(Java의 부품객체를 포함) 이 부품들을 잘 리용하는것이 중요하다.

1.4.6. 프로그람의 실행

코드작성을 끝낸 다음에는 프로그람을 콤파일, 련결, 집행하여야 한다. 【Run】차림표의 【Run Project】부분차림표를 선택하면 이 모든 공정을 완성할수 있다. 프로그람실행에서 만일 【Click Me】단추를 찰칵하거나 【New】차림표의 【Click Me】지령을 선택하면 그림 1-28에서 보여준 실행결과가 나타난다.



그림 1-28. 프로그람의 실행결과

1.4.7. 응용프로그람의 배비

응용프로그람작성이 끝나면 프로그람들을 배비하여야 한다. Archive Builder는 프로그람에서 요구하는 모든 파일들을 수집하여 이 파일들을 JAR파일로써 보존할수 있다. 이

것은 다음과 같이 실현할수 있다.【File】차림표의【New】지령을 선택하면【Object Gallery】대화칸이 현시된다.

여기서 【Archive】항목을 선택하고 그림 1-29와 같이 【Application】아이콘을 선택한다. 【OK】단추를 찰칵하면 【Archive Builder】대화칸이 나타난다.(그림 1-30)

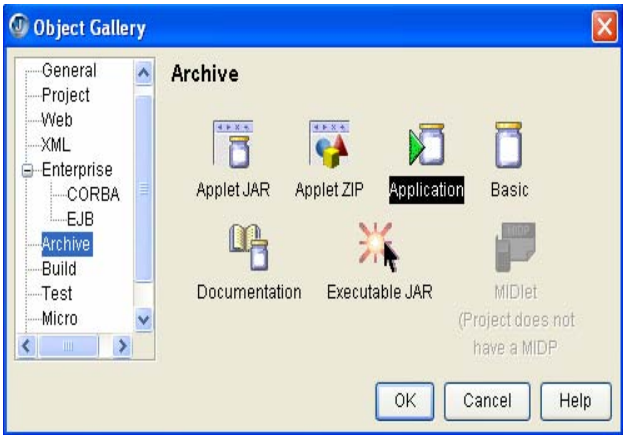


그림 1-29. Archive항목선택

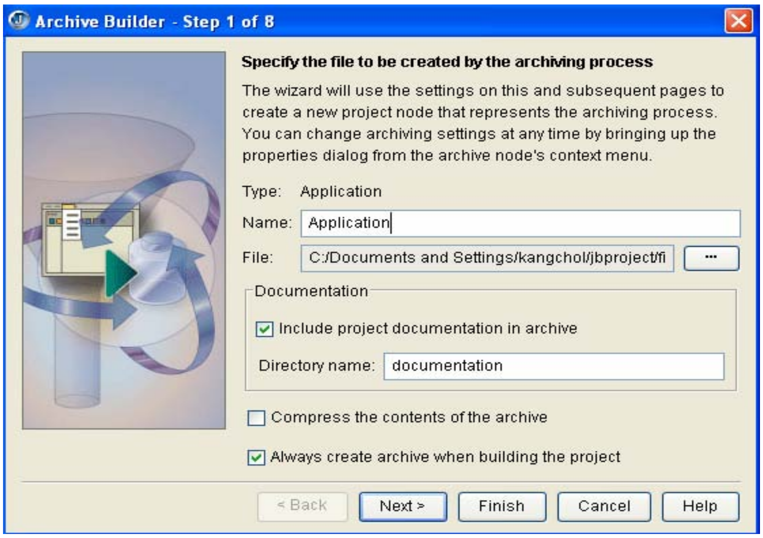


그림 1-30. Archive Builder조수대화칸의 첫 페지

첫 페지부터 4번째 페지까지는 기정값을 리용하고 5번째 페지에서는 Application추가선택항목을 그림 1-31과 같이 선택한다.

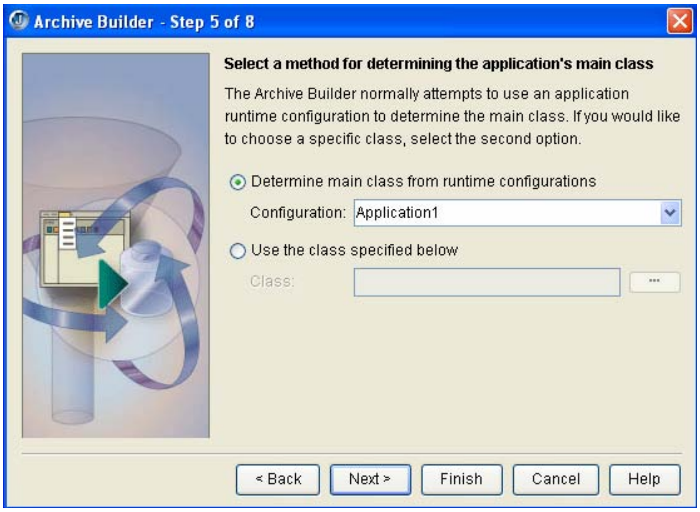


그림 1-31. Archive Builder조수대화칸의 5번째 페지

【Finish】단추를 찰칵하여 【Archive Builder】대화칸을 닫는다. 프로젝트란에 이름이 Application인 마디점이 나타나는데 마우스오른쪽단추를 찰칵하고 Properties를 선택하여 이 파일들을 수정할수 있다.(그림 1-32)

【Project】차림표의 【Make Project】지령을 선택하여 응용프로그람을 콤파일하고 JAR파일을 창조한다. Archive Builder는 프로젝트등록부의 모든 파일들을 JAR파일로 수집하고 Application보존마디점과 서로 린접한 아이콘을 펼치여 firstapp.jar보존파일을 조사한다. 프로젝트란에서 JAR파일을 두번 찰칵하면 적재부(manifest)파일이 Content페지에 나타나며 JAR파일내용은 그림 1-32와 같이 구조판에 현시된다.

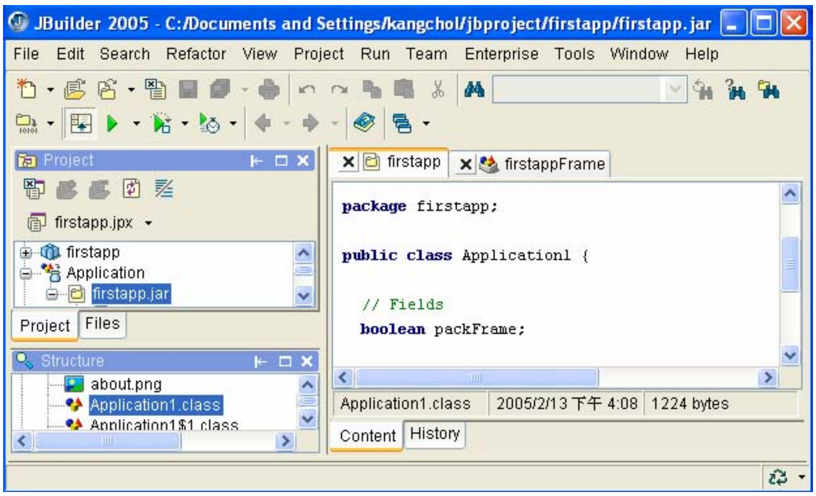


그림 1-32. JAR파일의 찾아보기

1.4.8. 응용프로그람고유코드의 생성

JBuilder에서 실행하는 응용프로그람외에 Windows조작체계에서 직접 실행할수 있는 고유응용프로그람을 작성할수 있다. 즉 Windows console응용프로그람, Linux체계, Solaris체계, Mac OS에서의 고유응용프로그람들을 생성할수 있다. 그 방법은 다음과 같다.

우선 【File】차림표의 【New】지령을 선택하여 그림 1-33과 같이 Native Executable Builder를 위한 조수대화칸을 연다.

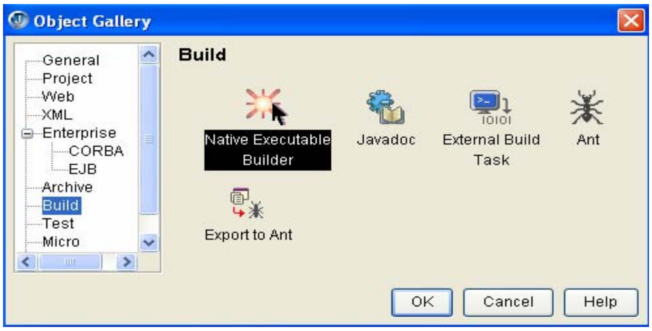


그림 1-33. Native Executable Builder를 위한 대화칸

1페지부터 6페지까지는 기정값으로 설정하고 그림 1-34에서와 같이 7페지에서 《Windows GUI "firstapp2W.exe"》검사칸을 선택한다. 그리하여 Windows조작체계에서 직접 실행할수 있는 고유응용프로그람을 창조한다.

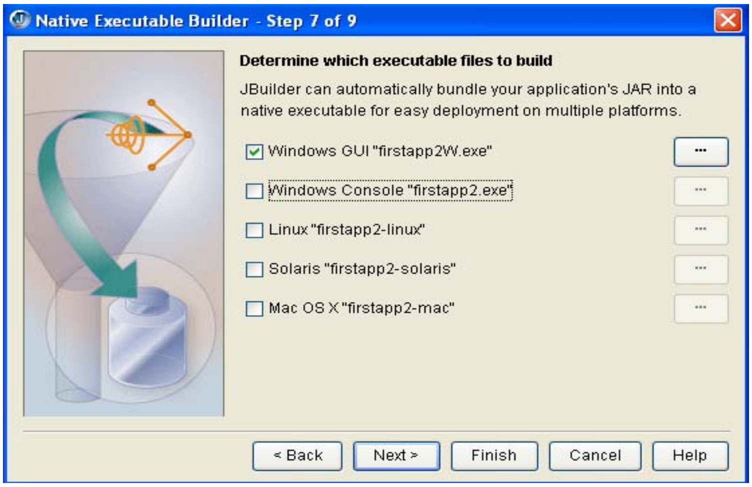


그림 1-34. Native Executable Builder대화칸의 7번째 페지

보는바와 같이 Windows console응용프로그람, Linux체계, Solaris체계, Mac OS에서의 고유응용프로그람을 생성하려면 여기의 검사칸을 선택하여 이 가동기반들에서의 고유코드를 생성하면 된다.

Java의 다중기반에 대한 개발환경은 꼭 같지 않으며 속도 역시 차이난다. 그러나 한번 작성하면 여러 기반에서 Java언어를 실행할수 있다. 【Finish】단추를 찰칵하여 모든 조작을 끝낸다. 그러면 프로젝트판에 Native Executable항목이 추가된다. 이 항목을 선택한 상태에서 마우스오른쪽단추를 찰칵하여 지름차림표를 연다. 여기서 【Make】항목을 선택하여 고유응용프로그람을 생성한다. 그밖에 실제상 firstapp2.jar파일이 더 생기는데 이것은 Archive Builder기능에 의하여 작성한것과 같다. 따라서 Native Executable Builder만 사용하면 그 기능뿐아니라 Archive Builder기능도 함께 수행된다.

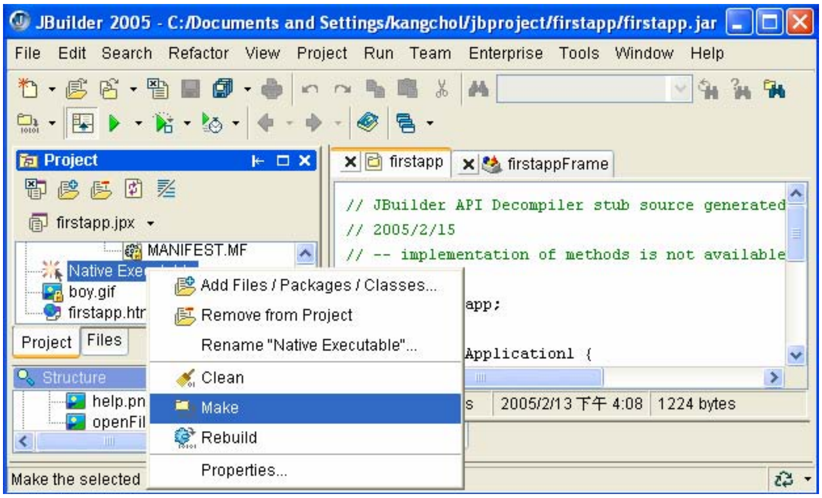


그림 1-35. Native Executable파일의 생성

제5절. 간단한 Java응용프로그람의 작성

아래에서 JBuilder를 사용하여 간단한 응용프로그람을 작성하는 실례를 고찰한다.

1.5.1. Java응용프로그람의 작성

JBuilder의 Application Wizard를 사용하여 새로운 응용프로그람을 작성하고 프로그람의 실행창문에 《Welcome to JBiulder 2005!》를 현시해보자.(그림 1-36)



그림 1-36. “Welcome” 을 실행한 현시결과

프로그람작성단계는 아래와 같다.

1) Project Wizard를 리용한 프로젝트의 창조

Project Wizard를 사용하여 이름이 *** .jpx인 프로젝트를 창조한다. 프로젝트파일은 프로젝트안의 기타 파일들이 들어있는 등록부정보를 포함하고있다. Project Wizard를 사용하는 구체적인 단계는 다음과 같다.

단계

1 【File】→【New Project】지령을 선택한다.

2 【Name:】본문칸에서 《untitled》를 《welcome》으로 수정한다.

3 【Directory:】복합칸에서 파일을 보관하려는 등록부를 선택한다.

4 이 페지에서 각종 환경설정을 하고 현재 사용할 값을 기정값으로 설정한다. 【Next】단추를 찰칵하여 다음 페지로 넘어간다.

5 【Encoding:】복합칸에서 《Big5》를 선택한다. 【Description:】마당에 《This is the first program》을 추가한다. 【Title:】마당에 《Welcome!》을 입력한다.

6 【Finish】단추를 찰칵한다.

2) Application Wizard의 사용

Application Wizard를 사용하는 구체적인 단계는 다음과 같다.

단계

1 【File】→【New】지령을 선택하면 New대화칸이 나타난다. 이 대화칸은 사용자가 각종 서로 다른 파일들을 창조하는데 쓰이는 조수대화칸이다.

2 Application아이콘을 찰칵하면 이때 Application Wizard대화칸이 나타난다.

3 【Class name:】본문칸에 《welcome》을 입력한다.

4 【Next】단추를 찰칵한다.

5 【Finish】단추를 찰칵한다. 이때 AppBrowser창문의 Navigation판에는 원천파일인 Frame1.java와 welcome.java이 현시된다. 임의의 파일을 선택한 다음 Source태브를 찰칵하면 그 파일의 내용이 내용판에 현시된다.

6 【Run】→【Run “welcome”】지령을 선택하거나 【F9】건을 누른다. 그러면 “welcome”창문이 나타난다.

7 Welcome응용프로그람을 닫는다.

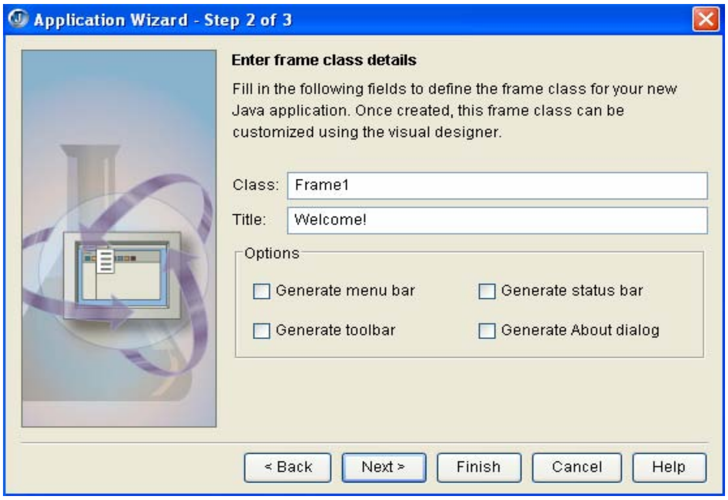


그림 1-37. Application Wizard대화칸의 2번째 페지

3) 프레임에 조종부품을 추가

프레임에 조종부품을 추가하는 구체적인 단계는 다음과 같다.

단계

1 프로젝트판에서 Frame1.java를 선택한다. 다음 Design태브를 찰칵한다.

2 부품선택판에서 AWT서고의 java.awt.Label부품을 선택한다.

3 Design방식에서 Label부품을 그린다.

4 Label의 text속성에 《Welcome to JBuilder 2005!》을 입력한다.

5 【File】→【Save All】지령을 선택하여 완성된 내용들을 보관한다.

6 【Run】→【Run "welcome"】지령을 선택하여 콤파일, 실행을 진행한다.(그림 1-36)

1.5.2. 실례에 대한 간단한 분석

이 프로그람의 원천코드(이 코드는 모두 JBuilder에 의하여 자동생성된것이다.)는 다음과 같다.

```
Frame1.java
package welcome;
import java.awt.*;
import javax.swing.*;
/*** <p>Title: welcome!</p>
```

```
 *
 * <p>Description: This is the first program </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: </p>
 *
 * @author kang chol
 * @version 1.0
 */
public class Frame1 extends JFrame {
    JPanel contentPane;
    BorderLayout borderLayout1 = new BorderLayout();
    Label label1 = new Label();
    public Frame1() {
        try {
            setDefaultCloseOperation(EXIT_ON_CLOSE);
            jbInit();
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
    /**
     * Component initialization.
     *
     * @throws java.lang.Exception
     */
    private void jbInit() throws Exception {
        contentPane = (JPanel) getContentPane();
        contentPane.setLayout(borderLayout1);
        setSize(new Dimension(400, 300));
        setTitle("Welcome!");

label1.setAlignment(Label.CENTER);
        label1.setFont(new java.awt.Font("Dialog", Font.BOLD, 20));
        label1.setText("Welcome to JBuilder 2005!");
```

```
        contentPane.add(label1, java.awt.BorderLayout.CENTER);
    }
}
Welcome.java:
package welcome;
    import java.awt.Toolkit;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import java.awt.Dimension;
/**
 * <p>Title: Welcome!</p>
 *
 * <p>Description: This is the first program </p>
 *
 * <p>Copyright: Copyright (c) 2005</p>
 *
 * <p>Company: </p>
 *
 * @author kang chol
 * @version 1.0
 */
public class Application1 {
    boolean packFrame = false;
    /**
     * Construct and show the application.
     */
    public Application1() {
        Frame1 frame = new Frame1();
        // Validate frames that have preset sizes
        // Pack frames that have useful preferred size info, e.g. from their
layout
        if (packFrame) {
            frame.pack();
        } else {
            frame.validate();
        }
```

```
        // Center the window
        Dimension screenSize =
                         Toolkit.getDefaultToolkit().getScreenSize();
        Dimension frameSize = frame.getSize();
        if (frameSize.height > screenSize.height) {
            frameSize.height = screenSize.height;
        }
        if (frameSize.WIDTH > screenSize.WIDTH) {
            frameSize.WIDTH = screenSize.WIDTH;
        }
        frame.setLocation((screenSize.WIDTH - frameSize.WIDTH) / 2,
                      (screenSize.height - frameSize.height) / 2);
        frame.setVisible(true);
    }
    /**
     * Application entry point.
     *
     * @param args String[]
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                try {
                    UIManager.setLookAndFeel(
                       UIManager.getSystemLookAndFeelClassName());
                } catch (Exception exception) {
                    exception.printStackTrace();
                }
                new Application1();
            }
        });
    }
}
```

이 응용프로그람은 2개의 파일을 가지고있다. 즉 Frame1.java와 Welcome.java파일이다.

매개 파일의 머리부에는 《package welcome;》이 있는데 이 명령문은 2개의 파일이 welcome패키지에 속한다는것을 의미한다. import명령문은 패키지를 반입하는 역할을 한다. 이것은 C언어에서의 #include명령문과 류사하다. 실례로

import java.awt.*;

이것은 Awt패키지를 반입한다는것이다.

class는 새로운 클라스를 선언하는데 쓰인다. 그 뒤에는 클라스이름을 쓴다. 실례로 《public class welcome》은 이름이 welcome인 클라스를 정의하고있다. Welcome.java에서의 클라스 welcome은 main()메쏘드를 정의하고있다. 즉

public static void main(String [] args)

여기서 public는 접근권한을 표시하며 static는 이 메쏘드가 클라스의 정적메쏘드이라는것을 가리킨다. void는 main()메쏘드가 아무런 값도 귀환하지 않는다는것을 가리킨다. 응용프로그람에서 main()메쏘드는 반드시 있어야 하며 우의 서식에 따라 정의하여야 한다. Java해석기는 main()을 입구로 하여 프로그람을 집행한다. Java프로그람에서는 여러개의 클라스를 정의할수 있고 매개 클라스에서는 여러개의 메쏘드들을 정의할수 있다. 그러나 적어도 하나는 공개클라스이여야 한다.

main()메쏘드에서 괄호안의 String args [] 는 main()메쏘드에 전달하는 파라메터이며 파라메터이름은 args이다. 이것은 클라스 String의 구체례(instance)이며 파라메터의 개수는 0개 또는 여러개일수도 있다. 매개 파라메터는 《클라스이름.파라메터이름》의 형식으로 지정하며 여러개의 파라메터들사이를 반점으로 구분한다. 그밖에 《.》은 객체성원에 대한 인용을 표시한다. 실례로

contentPane.setLayout(borderLayout1);

이렇게 하여 Java응용프로그람의 작성방법에 대하여 간단히 고찰하였다.

제2장. JBuilder의 통합개발환경(IDE)

JBuilder의 시각적인 통합개발환경은 본문편집기, 프로젝트작성도구, 구조판과 오유제거기 등을 포함하고있다. 사용자는 통합개발환경에서 프로젝트를 창조하거나 프로젝트와 파일들을 열고 편집할수 있다. 또한 응용프로그람을 콤파일하고 련결,실행, 오유제거를 할수 있다.

JBuilder의 통합개발환경은 그림 2-1과 같은 단일창문대면부로서 파일과 프로젝트의 편집과 관리, 시각적인 대면부설계, 열람, 콤파일, 오유제거 등의 여러가지 기능을 일체화한 개발기반이다.

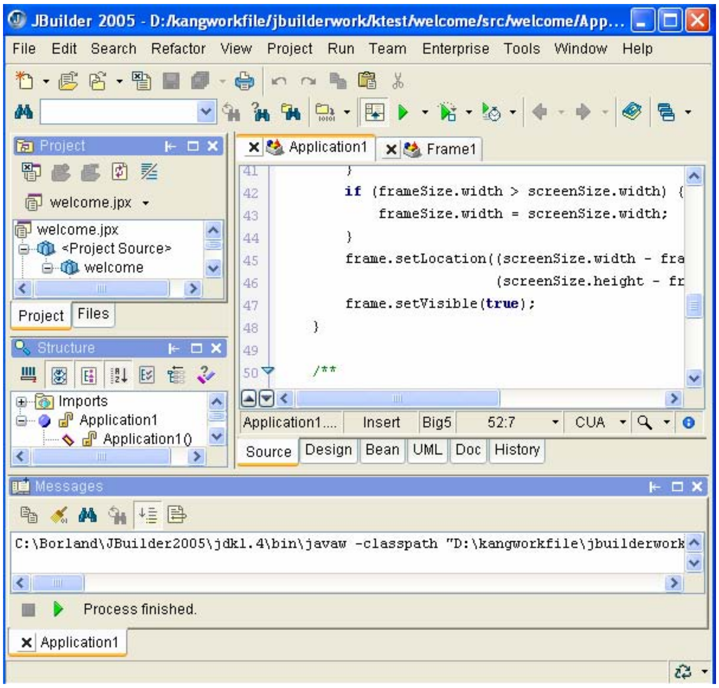


그림 2-1. JBuilder 의 통합개발환경

이 장에서는 JBuilder의 통합개발환경 (IDE; Integrated Development Environment)의 특징, JBuilder의 기본차림표, 기본대면부의 지름방식과 사용방법들을 중점적으로 설명하고 통합개발환경에서의 프로그람작성방법을 서술한다.

제1절. 기본차림표띠

차림표띠는 조작지령들을 부류별로 묶어놓은 여러개의 차림표들로 구성되여있으며 마우스나 지름건을 리용하여 선택할수 있다. 차림표를 찰칵하면 내리펼침차림표가 펼쳐진다. 만일 차림표항목의 지령을 집행하려면 상응한 차림표항목을 찰칵하면 된다. 또한 차림표항목의 오른쪽열에 지름건이름들이 있는데 건반에서 이 건을 누르면 해당한 지령이 실행된다.

아래에서 자주 사용하면서도 비교적 중요한 차림표들에 대하여 설명한다.

2.1.1. 파일차림표【File】

1. 【New...】지령

【File】→【New...】를 선택하면 Object Gallery대화칸이 나타난다.(그림 2-2) 이 대화칸에서 파일, 프로젝트, 응용프로그람, 웨브응용프로그람, XML응용프로그람, CORBA, Build, Enterprise응용프로그람 등을 창조할수 있다.

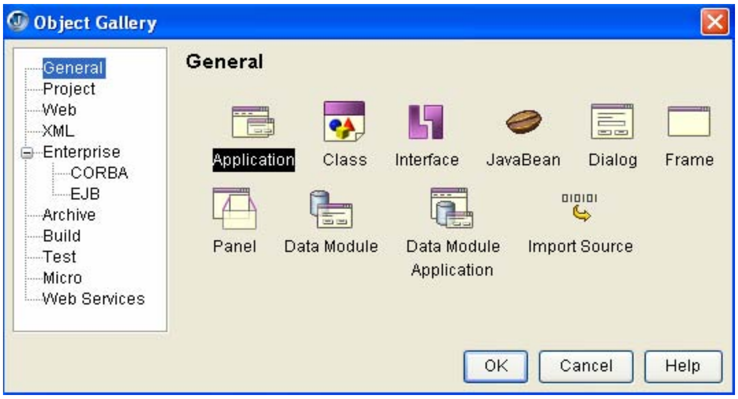


그림 2-2. Application아이콘의 선택

이 대화칸에는 10개의 항목이 있는데 아래에서는 주요 항목들에 대하여 설명한다.

- General항목

General항목에서는 여러가지 류형의 파일들을 창조할수 있다. 어떤 파일을 창조하려면 상응한 아이콘을 두번 찰칵한다. General항목에 속하는 항목류형은 표 2-1과 같다.

표 2-1. General에 속하는 항목들

항목류형	설 명
Application	프레임을 가진 응용프로그람을 창조한다.
Class	프로젝트에서 객체를 창조한다. 이것을 사용하면 자기의 객체클라스를 정의할수 있다.

Interface	대면을 창조한다.
JavaBean	JavaBean 부품클라스를 창조한다.
Dialog	의뢰기측 Dialog 객체클라스를 창조한다.
Frame	Frame 객체클라스를 창조한다. 이 객체클라스를 사용하면 Frame 객체클라스의 입력과 그리기기능을 정의하고 확장할수 있다.
Panel	Panel 객체클라스를 창조한다. 이 클라스는 용기클라스로서 부분객체메쏘드를 덧태우기하여 중복그리기 등의 기능을 실현한다.
Data Module	Data Module 객체를 창조하고 볼수 없는 자료를 파일과 련결한다.
Data Module Application	기정의 Data Module을 사용하는 응용프로그람을 창조한다.

- Project항목

여기서는 새로운 프로젝트를 창조할수 있다. 매 항목들의 내용은 표 2-2에서 서술한다.

표 2-2. Project에 속하는 항목들

항 목 류 형	설 명
Project	프로젝트를 창조한다.
Project For Existing Code	기존프로젝트로부터 하나의 프로젝트를 창조한다.
Pull Project From CVS	프로젝트를 CVS와 련결하고 고유프로젝트를 배비한다.
Pull Project From VSS	프로젝트를 Visual SourceSafe와 련결하고 원격자료기지를 가지고 고유프로젝트를 배비한다.
Pull Project From ClearCase	프로젝트를 ClearCase와 련결하고 선택한 VOB를 결정한다.
Pull Project From StarTeam	프로젝트를 StarTeam과 련결하고 고유프로젝트를 배비한다.

- Web항목

이 항목에서는 웨브와 관련있는 응용프로그람들과 파일들을 창조한다.(표 2-3)

표 2-3. Web에 속하는 항목들

항 목 류 형	설 명
Applet	애플레트의 Internet Java Applet 소응용프로그람을 창조한다.
Web Application	웨브응용프로그람을 창조한다.
Servlet	Javax.servlet.http.HttpServlet 를 확장한 Java 파일을 창조한다.
Java Server Page	JSP 의 프레임을 창조한다.
Web Start Launcher	JNLP 파일과 응용프로그람의 HTML 파일을 생성한다.

- XML항목

이 항목에서는 XML과 관련있는 응용프로그람과 파일들을 창조한다.(표 2-4)

표 2-4. XML에 속하는 항목들

항 목 류 형	설 명
Cocoon Web Application	Cocoon Web Application 응용프로그람을 창조한다.
Databinding	DTD 나 XSD 파일로부터 Java 객체를 창조한다.
DTD To XML	DTD 로부터 XML 파일을 창조한다.
XML To DTD	XML 로부터 DTD 파일을 창조한다.
XML-DBMS	DTD 로부터 Map 파일과 SQL 파일을 생성한다.

- CORBA항목

이 항목에서는 CORBA와 관련이 있는 여러 층의 응용프로그람에 대한 봉사기/의뢰기의 대면과 프로그람을 창조한다.(표 2-5)

표 2-5. CORBA 에 속하는 항목들

항 목 류 형	설 명
Sample IDL	IDL견본프로그람파일을 창조한다.
CORBA Client Interface	CORBA봉사기측대면에 접근하는 JavaCORBA 의뢰기측객체를 창조한다.
CORBA Server Interface	IDL파일로부터 CORBA봉사기측대면객체를 창조한다.
HTML CORBA Client	IDL파일로부터 HTML CORBA의뢰기측응용프로그람을 창조한다.
CORBA Server Application	IDL파일로부터 CORBA봉사기측응용프로그람의 객체를 창조한다.

- Enterprise항목

이 항목에서는 Enterprise응용프로그람과 관련있는 대면과 객체를 창조할수 있다. 이 기능은 기업판의 JBuilder에만 있다.(표 2-6)

표 2-6. Enterprise 에 속하는 항목들

항 목 류 형	설 명
EJB Module	자기의 Enterprise beans를 추가할수 있는 EJB모듈을 창조한다. EJB모듈은 간단히 말하면 jar파일에서 enterprise beans를 배비하는데 쓰이는 론리적인 모임이다.
EJB Module From Descriptors	Enterprise beans를 배비하는데 쓰이는 서술자파일로부터 새로운 EJB모듈을 창조한다.
E.terprise JavaBean 1.x	Enterprise Java Bean 1.x부품객체를 창조한다.
EJB 1.x Entity Bean Modeler	기존의 자료기지표로부터 하나이상의 enterprise beans 1.x부품을 창조한다.
EJB 2.0 Bean Designer	EJB모듈을 선택하거나 필요하면 EJB모듈을 창조하며 다음에 EJB Designer를 현시하여 EJB 2.0부품객체에 대한 시각적인 설계를 한다.

항 목 류 형	설 명
EJB Test Client	간단한 의뢰기측응용프로그람을 창조할수 있다. Enterprise bean부품객체의 기능을 검사하는데 쓰인다.
EAR	Enterprise Archive(EAR)파일을 창조한다.
JMS	Java Message Service(JMS)체계에서 통보문을 지원하는 생산자와 소비자의 코드를 가지고있는 Java클라스를 창조한다.

- Builder항목

이 항목에서는 코드생성 및 고유코드생성, 프로젝트창조과정에서 사용되는 과제를 창조한다.(표 2-7)

표 2-7. Builder 항목에 속하는 항목들

항 목 류 형	설 명
Archive Builder	모든 항목과 포함하여야 할 패키지들이 있는 jar나 zip형의 파일들을 창조한다.
Native Executable Builder	고유집행할수 있는 JAR응용프로그람파일을 자동적으로 창조한다.
Javadoc wizard	API원천파일의 해설문으로부터 Javadoc를 창조한다.
External Build Task	프로젝트창조시 집행하는 확장된 과제를 창조한다.

- Test항목

이 항목에서는 시험에 필요한 응용프로그람프레임과 클라스객체를 창조할수 있다. (표 2-8)

표 2-8. Test 에 속하는 항목들

항 목 류 형	설 명
Test Case	Junit.framework.TestCase를 확장하고 프로젝트에서 클라스객체를 검사할수 있는 주요 메쏘드의 TestCase클라스프레임을 창조한다. 이 프레임은 setUp()과 tearDown(), 그리고 검사하는데 필요한 몇가지 기타 메쏘드들을 포함하고있다.

Test Suite	Test Suite조수에서는 test cases로부터 test suite까지에서 선택하며 묶음처리실행과 같이 test suite클라스의 객체를 생성한다.
JDBC Fixture	JDBC Fixture클라스를 창조한다.
JNDI Fixture	JNDI Fixture클라스를 창조한다. 단위검사에서 JNDI에 대한 검색결과를 현시한다.
Comparison Fixture	Comparison Fixture클라스를 창조한다. 검사출력을 기록하고 현재의 검사출력과 이전의 출력기록을 비교한다.
Custom Fixture	setUp()과 tearDown()메쏘드를 포함하고있는 test fixture클라스의 프레임을 창조한다.

2. 【New Project】지령

【File】→【New Project】지령은 JBuilder프로젝트파일을 창조한다. 이 지령을 찰칵하면 표준적인 JBuilder프로젝트파일을 창조하는 대화칸이 나타난다. 여기서 가리키는 항목에 따라 JBuilder프로젝트파일을 창조할수 있다.

3. 【New Class】지령

【File】→【New Class】지령은 Java class객체를 창조한다. 이 지령을 찰칵하면 표준적인 Java class객체를 창조하는 대화칸이 나타난다. 여기의 항목에 따라 Java class객체파일을 창조할수 있다.

4. 【Open Project】지령

【File】→【Open Project】지령은 이미 있는 프로젝트를 연다. Delphi와 다른 점은 JBuilder는 여러개의 프로젝트들을 동시에 열수 있으며 또한 여러개의 응용프로그람열람기를 열어 서로 다른 프로젝트들을 각각 현시할수 있다. 이 차림표의 조작방법은 【File】→【Open File】지령에 대한 조작방법과 완전히 같다.

5. 【Open File】지령

이 지령의 지름건은 【Ctrl+O】이며 이것을 리용하여 이미 존재하는 JBuilder파일, JBuilder프로젝트파일, JBuilder패키지, C와 C++파일, SQL파일, HTML파일, 묶음처리파일, 본문파일 등을 열수 있다. 이 지령을 찰칵하면 표준적인 파일열기대화칸이 나타나며 이 대화칸에서 서로 다른 위치에 있는 파일들을 선택할수 있다. 만일 파일이 프로젝트파일이면 통합개발환경에서 열린다. 기타 파일들은 코드편집기에서 파일을 찾아 그것을 편집할수 있다.

6. 【Reopen】지령

【File】→【Reopen】지령은 최근에 사용한 프로젝트나 모듈을 연다. JBuilder는 최근에 연 프로젝트나 프로젝트파일에 대하여 기억기능을 가지고있다.

7. 【Compare Files...】지령

【File】→【Compare Files...】은 파일을 비교하는 조작을 한다.

8. 【Close Projects...】지령

이것은 현재 화면에 현시된 프로젝트파일들을 닫는다. 만일 현재 파일이 변경되였으면 수정한것을 보관하겠는가를 물어보는 대화칸이 현시된다.

9. 【Close “firstappFrame.java”】지령

이 지령은 현재 화면에 현시된 java파일을 닫는다. 만일 현재 파일이 수정되였으면 대화칸이 나타나는데 이것은 수정한것을 보관하겠는가를 물어보는 대화칸이다. 이 차림표의 지름건은 【Ctrl+F4】이다.

10. 【Close All Except “firstappFrame.java”】지령

이 지령은 현재 화면에 현시된 프로젝트파일을 제외한 나머지 프로젝트파일들을 모두 닫는다. 만일 닫으려는 파일이 변경되였으면 보관대화칸이 나타난다.

11. 【Close...】지령

이 지령은 닫기대화칸을 현시한다. 이 지령의 지름건은 【Ctrl+Shift+F4】이다.

12. 【Revert “firstappFrame.java”】지령

이것은 현재 화면에 현시된 java파일을 원상대로 회복한다. 만일 현재 파일이 변경되였으면 수정전의 파일로 회복하는 기능을 수행한다. 이 차림표는 현재 파일이 변경되였을 때에야 능동으로 된다.

13. 【Save Project “firstapp.jpx”】지령

이것은 현재 화면에 현시된 프로젝트파일들을 보관한다.

14. 【Save Project As】지령

이 지령은 프로젝트파일들을 다른 이름의 파일로 보관한다. 이 조작방법과 기능은 【Save As】지령의 사용방법과 같다.

15. 【Save “firstappFrame.java”】지령

현재의 파일을 보관하는 지령인데 지름건으로는 【Ctrl+S】이다. 파일류형은 프로젝트파일이나 다른 류형의 파일들이다. 만일 파일이 이미 존재하면 자동적으로 원래의 파일이름과 경로에 따라 파일을 갱신하고 보관한다. 그렇지 않으면 표준적인 파일보관대화칸이 열리는데 여기에 새로운 파일이름과 경로를 입력할수 있다.

16. 【Save As】지령

이 지령은 현재의 파일을 새로운 파일로 보관한다. 이 지령을 선택하면 표준적인 파일보관대화칸이 나타나는데 사용자는 이 대화칸에서 파일이름과 파일경로를 입력할수 있다. 만일 현재 파일이 보관된것이 없다면 그의 기능은 【Save】지령과 같다.

17. 【Save All】지령

JBuilder통합개발환경에서 열었던 모든 파일들을 보관하는 지령인데 이것의 지름건은 【Ctrl+Shift+A】이다. 만일 파일이 이미 존재하면 자동적으로 원래의 파일이름과 경로에 따라 파일을 갱신보관한다. 그렇지 않으면 표준적인 파일보관대화칸이 열리는데 여기에 파일이름과 파일의 존재경로를 입력할수 있다.

18. 【Rename "firstappFrame.java"】지령

이 지령은 현재 파일이름을 다른 이름으로 바꾼다.

19. 【Page Layout...】지령

이 지령은 페지에 대한 설정을 진행한다. 이 지령을 사용하면 페지설정대화칸을 열어 각종 페지속성들을 변경할수 있다.

20. 【Print】지령

이 지령을 실행하면 표준적인 인쇄대화칸이 열리며 코드편집기의 프로그람코드와 설계창문을 인쇄할수 있다.

21. 【Exit】지령

이 지령은 Jbuilder를 완전히 닫는다.

2.1.2. 편집차림표 【Edit】

편집차림표는 설계단계에서의 본문과 부품을 관리하는데 쓰인다. 표 2-9에서는 편집에서 중요하게 쓰이는 CodeInsight의 지령들에 대하여 설명한다. 여기서 일반지령들인 자르기, 복사, 붙이기 등은 설명하지 않는다.

표 2-9. CodeInsight의 주요 지령

지　　령	설　　명
MemberInsight	접근할수 있는 자료성원과 메쏘드를 현시한다.
ParameterInsight	코드편집중에 있는 메쏘드의 파라메터표현식을 현시한다.
ClassInsight	현재 클라스경로상에서 접근할수 있는 클라스목록을 현시한다.

표 2-9로부터 알수 있는바와 같이 프로그람을 작성할 때에 【MemberInsight】,【ParameterInsight】와 【ClassInsight】는 필요한 정보들을 현시한다. 즉 자료성원과 메쏘드의 접근형식, 코드작성시 메쏘드의 파라메터표현식과 현재 클라스경로상에서 접근할수 있는 클라스목록 등을 현시한다. 조작방법은 다음과 같다. 마우스지시자를 정보를 현시하려는 클라스성원에까지 이동하고 다음에 차림표지령을 통하여 요구하는 결과를 얻는다. 그림 2-3은 【MemberInsight】지령의 사용결과이다.

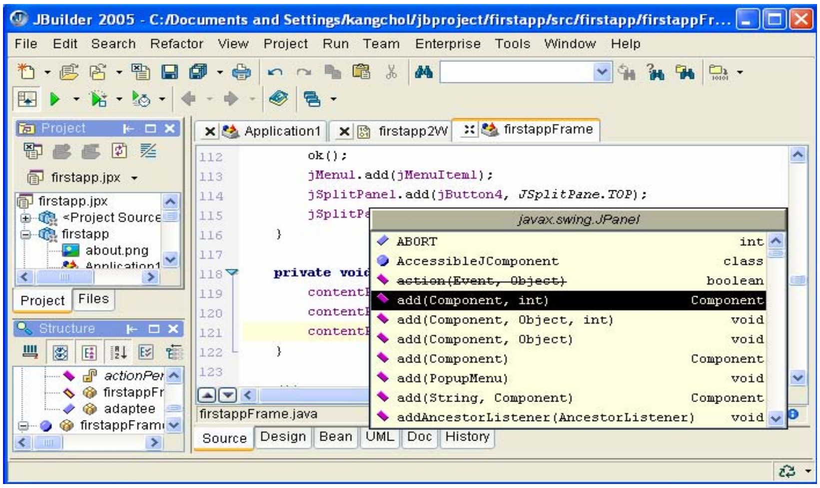


그림 2-3. 【MemberInsight】지령의 사용결과

2.1.3. 탐색차림표 【Search】

【Search】 차림표의 대표적인 지령들은 다음과 같다. 【Find】, 【Find in path】, 【Replace】, 【 Replace In path】, 【Search Again】, 【Incremental Search】, 【Go To Line】, 【Find Class】, 【Find Definition】, 【Find References】 등이 있다.

여기서 【Find】, 【Find in Path】, 【Replace】, 【Replace In path】, 【Search Again】, 【Incremental Search】와 【Go To Line】 등은 모두 일반적으로 쓰는 차림표지령들이다.

주의할것은 【Find Classes】지령이다. 【Find Classes】지령을 선택하였을 때 Find Classes대화칸이 나타난다.(그림 2-4) 이 대화칸에서 검색하려는 대상을 입력한다.

그리고 【Find Definition】과 【Find References】는 같은 효과를 가진다.

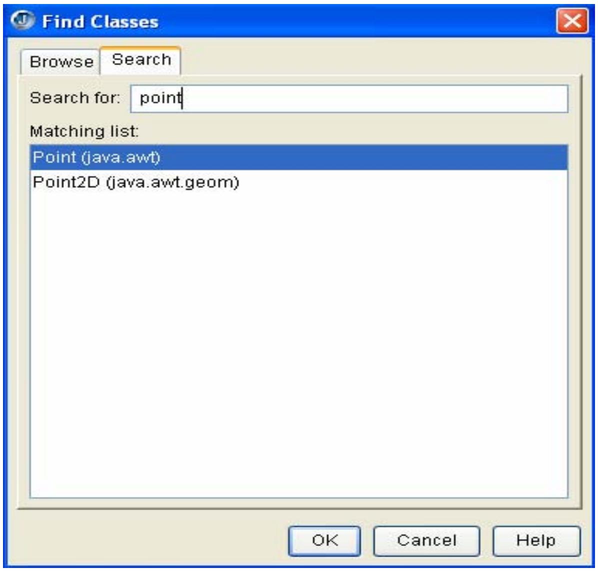


그림 2-4. Find classes대화칸

2.1.4. 재구성차림표【Refactor】

이 차림표는 사용자의 프로그람을 재구성(refactoring)하고 최적화하는 지령들을 포함하고있다. 여기에는 【Refactor】, 【Refactoring History】, 【Distributed Refactorings】,【Optimize Imports】,【Rename】,【Move】,【Inline】,【Change Parameters】, 【Extract Interface From】, 【Introduce Superclass For】, 【Pull Up】, 【Push Down】, 【Extract Method】, 【Introduce Variable】, 【Introduce Field】, 【Surround with Try/Catch】, 【Introduce Foreach】, 【Introduce Auto(un)boxing】,【Introduce Generics】 지령들이 있다. 아래에서 기본적인 지령들을 소개한다.

1. 【Refactor】지령

이 지령은 선택된 코드에 적합한 재구성을 진행한다.

2. 【Refactoring History】지령

이 지령은 Refactoring History대화칸을 현시하여 열려진 프로젝트나 서고보존파일에 추가된 재구성들을 개괄한다.

3. 【Distributed Refactorings】지령

이 지령은 Distributed Refactorings대화칸을 현시하여 여러 파일들의 재구성들을 볼수 있도록 한다. 재구성들은 미정 혹은 완결 재구성으로 일람표화된다.

미정재구성은 현재의 프로젝트나 프로젝트그룹에 아직 적용되지 않은것들로서 JAR나 서고에 대한 재구성이 미정목록에 현시된다.

완결재구성은 적용된것들을 말한다.

4. 【Optimize Imports】지령

이 지령은 import명령문을 전용프로젝트설정들에 따라 다시 쓰고 재구성하는데 쓰인다. 이것은 더는 사용할 필요가 없는 import명령문들을 제거할수도 있다. Project Properties Formating의 imports page에서 반입순서를 설정한다.

5. 【Rename】지령

이 지령은 패키지, 클라스, 내부클라스, 대면, 메쏘드, 마당, 국부변수, 속성 등의 이름바꾸기에 쓰인다.

6. 【Move】지령

지정된 클라스를 다른 패키지(이미 있는 패키지 또는 새로운 패키지)에로 이동시키는데 쓰인다. 이동재구성(Move refactoring)은 웃준위공개클라스를 현재 준위(마우스유표가 있는)로 이동할 때에만 리용한다. 클라스가 이동되는 패키지는 같은 이름을 가진 원천파일을 미리 포함할수 없다.

7. 【inline】지령

이 지령은 메쏘드나 변수를 직접삽입(inline)할 때 리용한다.

8. 【Change Parameters】지령

이 지령은 메쏘드파라메터들을 추가, 삭제, 재정렬할 때 리용한다.

9. 【Extract Method】지령

이 지령은 선택한 코드부분을 어떤 메쏘드에 삽입하여 넣을 때 리용한다.

10. 【Introduce Variable】지령

어떤 복합표현식이나 그것의 어떤 부분에 대한 결과를 림시변수이름을 리용하여 바꾼다.

11. 【Surround With Try/Catch】지령

이 차림표지령은 선정된 코드묶음에 try/catch명령을 추가할 때 리용한다.

우에서 서술한 지령들은 코드입력속도를 높일수 있게 한다. 실례로 그림 2-5에서 보여주는 푸른색부분의 코드를 선택하고 다음에 【Surround With Try/Catch】지령을 찰칵하면 JBuilder는 자동적으로 try/catch코드를 추가하여 표현식의 례외를 자동적으로 검측하도록 한다.

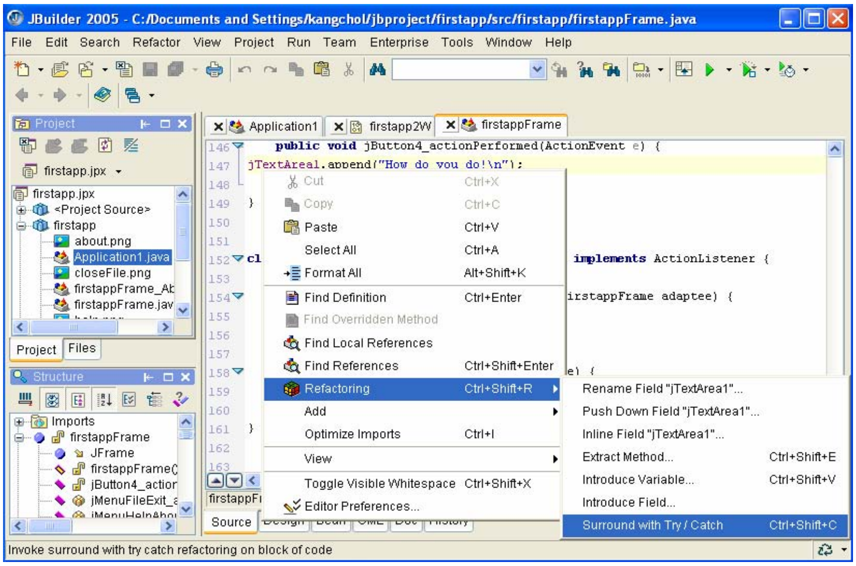


그림 2-5. 【Surround With Try/Catch】지령의 기능

12. 【Rename】지령

Rename은 JBuilder에서 선정한 패키지, 클라스, 메쏘드, 국부변수, 마당과 속성 등의 이름을 바꿀 때 리용한다. 그림 2-5의 jTextArea1를 jTextAreaNew1로 이름을 바꾸자면 다음과 같이 한다.

우선 jTextArea1변수를 선택한 다음 【Refactor】차림표의 【Rename Field "jTextArea1"】지령을 선택한다. 이때 그림 2-6과 같은 대화칸이 나타나는데 이 대화칸에서 수정할 이름 jTextAreaNew1을 입력하고 OK단추를 찰칵한다. 이때 Refactoring 대화칸이 열리는데 체계가 검측한 jTextArea1변수를 사용하는 모든 코드들이 렬거된다. 여기서 립방체단추를 찰칵하면 JBuilder는 모든 jTextArea1변수를 jTextAreaNew1로 이름을 바꾼다. 복원단추를 찰칵하면 체계는 jTextAreaNew1변수를 jTextArea1변수로 다시 회복한다.(그림 2-7)

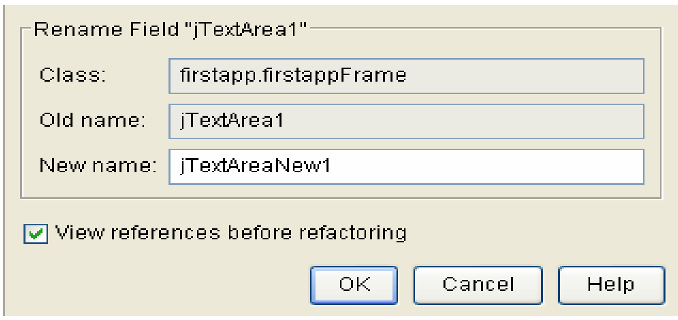


그림 2-6. Rename Field "jTextArea1" 대화칸

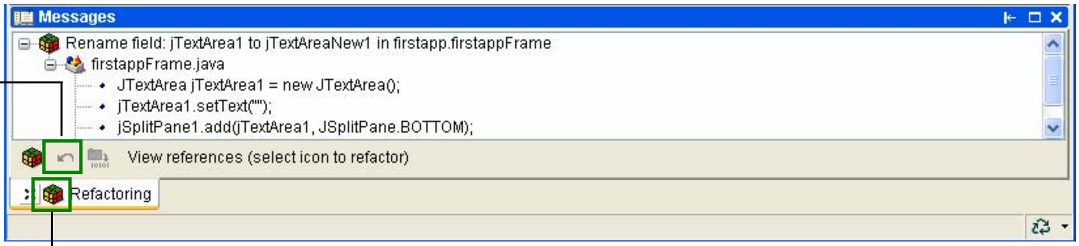


그림 2-7. Refactoring창문

13. 【Extract Method】지령

만일 어떤 메쏘드에서 코드가 너무 길다고 느껴지면 그것들을 여러개의 작은 메쏘드들로 가를수 있다. 【Extract Method】지령을 사용하면 선택한 코드를 어떤 메쏘드로 전환할수 있다. 실례로 임의의 코드를 선택한 다음 【Refactor】차림표의 【Extract Method】지령을 찰칵한다. 이때 나타나는 대화칸에서 Method name:본문칸에 《ok》를 입력하고 다시 【OK】단추를 찰칵한다.(그림 2-8) 그러면 선택한 코드가 ok메쏘드에 추가된다.(그림 2-9)

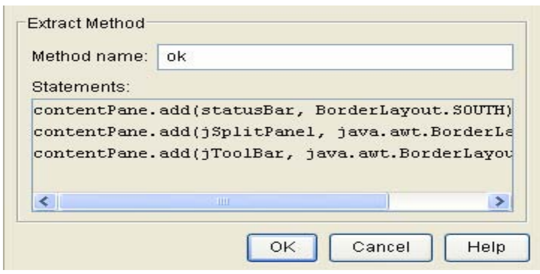


그림 2-8. Extract Method대화칸

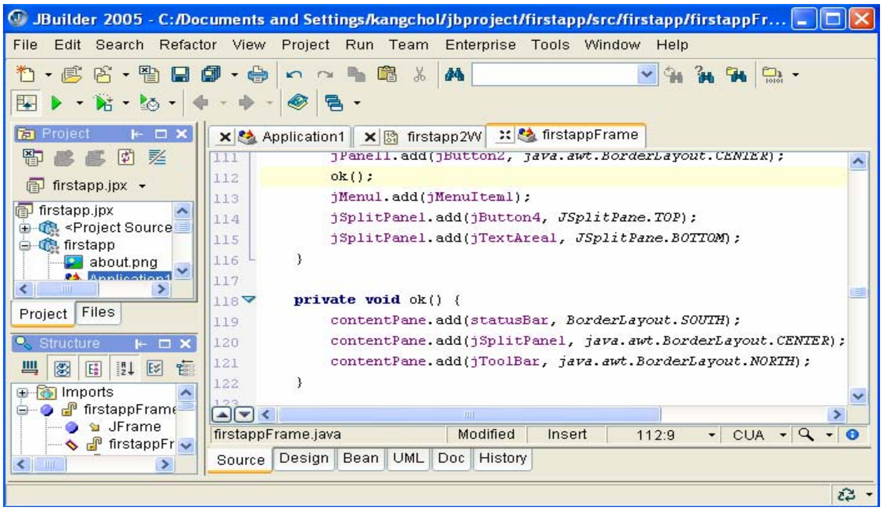


그림 2-9. 【Extract Method】지령집행후의 결과

이 방법은 프로그람의 작성속도와 코드입력속도를 높인다. 이와 류사한 지령으로서 【Introduce Variable】이 있다. 그의 사용방법은 【Extract Method】지령과 기본적으로 같으며 단지 표현식이나 부분표현식의 의미를 해석할수 있는 림시변수이름을 사용하였다는 것이다.

2.1.5. 보기차림표 【View】

보기차림표는 IDE창문에 어느 내용을 현시하겠는가를 결정할 때 리용한다. 이 차림표의 지령을 리용하여 사용자의 요구에 따라 프로그람작성환경을 바꿀수 있다.

보기차림표에서는 많은 중요한 도구들을 제공하고있다. 례하면 【ToolBars】(도구띠), 【Project】(프로젝트창문의 절환현시), 【Content】(내용창문의 절환현시), 【Structure】(구조창문 절환현시), 【States Bar】(상태띠 절환현시)와 【Hide All】 (모든 창문의 숨기기) 등이 있다. 【ToolBars】에서 현시하려는 도구를 선택할수 있다. 기본적으로 【File】, 【Editing】, 【Search】, 【Build】, 【Run/Debug】, 【Navigation】과 【Help】 등이 있다. 이 도구들은 도구띠에서 구체적으로 소개한다.

2.1.6 프로젝트차림표 【Project】

【Project】차림표에는 프로젝트를 처리하는데 필요한 차림표지령들이 있는데 기본적으로 다음과 같다.

- 【Make Project】지령은 현재 프로젝트의 모든 파일들을 콤파일하고 련결시킨다.
- 【Rebuild Project】지령은 현재 프로젝트의 모든 파일들을 다시 콤파일하고 련결시킨다.
- 【Make】지령은 현재 프로젝트의 현재 파일들을 콤파일한다.
- 【Rebuild】지령은 현재 프로젝트의 현재 파일을 다시 콤파일하고 련결시킨다.

- 【Apply Filter】지령은 려과기를 선택된 패키지들에 적용한다.
- 【New Folder】지령은 현재 프로젝트에 새로운 등록부를 추가한다.
- 【Remove from Project】지령은 현재 프로젝트에서 파일을 제거한다.
- 【Refresh】지령은 현재 프로젝트창문의 내용을 재생한다.
- 【Rename】지령은 현재 프로젝트파일의 이름을 바꾼다.
- 【Project Properties】지령은 현재 프로젝트의 속성을 설정한다.
- 【Default Project Properties】지령은 프로젝트속성을 기정으로 설정한다.

여기서 【Make Project】지령, 【Rebuild Project】지령, 【Make】지령과 【Rebuild】지령은 모두 보통 많이 쓰는 프로젝트차림표지령이다.

【Add Files/Packages】지령은 JBuilder자체가 자동적으로 창조하거나 추가할수 없는 파일을 추가한다.

실례로 현재의 프로젝트에 한개의 화상파일을 추가하는 방법에 대하여 보기로 하자. 【Project】차림표의 【Add Files/Packages】지령을 찰칵하면 《Add to "first app.jpx"》대화칸이 현시된다.(그림 2-10)

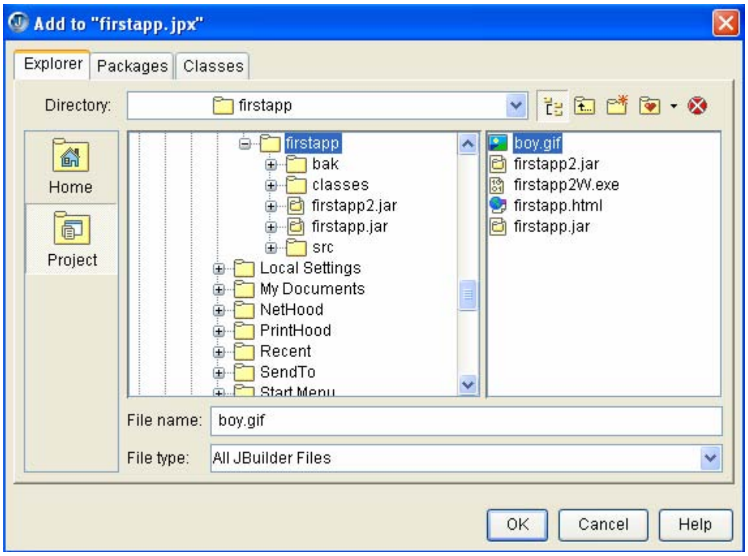


그림 2-10. Add to "firstapp.jpx" 대화칸

추가하려는 파일 boy.gif를 선택하고 【OK】단추를 찰칵하면 파일이 현재의 프로젝트에 추가된다. 만일 현재 프로젝트의 속성을 수정하려면 【Project Properties】지령을 찰칵하면 된다. 이때 Project Properties대화칸이 나타나는데 여기서 프로젝트를 창조할 때에 설정한 각종 속성파라메터들을 수정할수 있다.(그림 2-11) 이 대화칸에서 실례로 JDK

판본, 출력경로, 작업등록부 등 경로파라메터와 문자모임, 프로젝트의 표제, 묘사, 저자, 회사, 판본, 판권정보 등의 설정을 할수 있다.(그림 2-12) 이 대화칸의 Run페지는 현재 프로젝트의 주실행프로그람을 선택하는데 쓰인다.

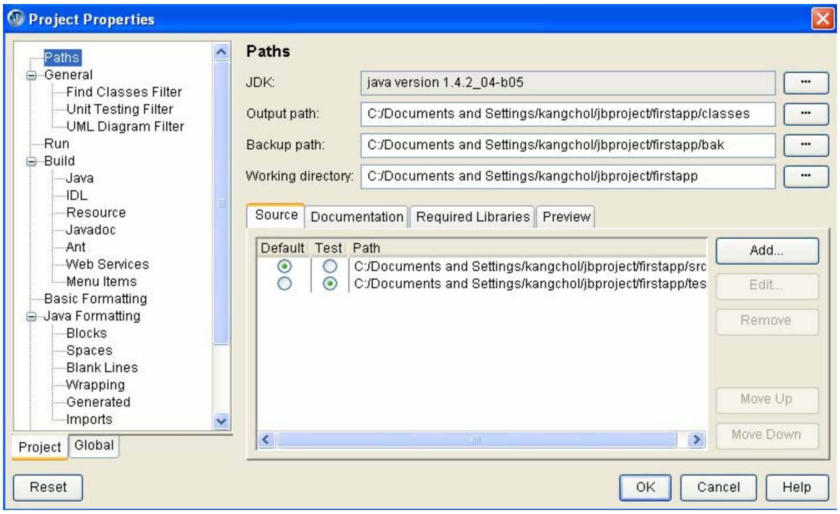


그림 2-11. Project Properties대화칸의 Paths페지

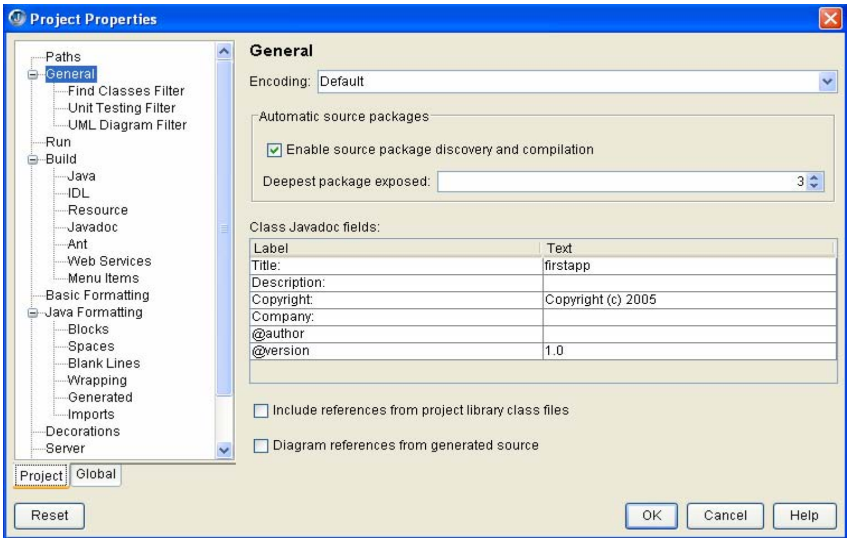


그림 2-12. Project Properties대화칸의 General페지

Project Properties대화칸의 Build는 Java, IDL, Resource, Javadoc, Ant, Menu Items와 Web Services인 7개의 페지를 포함하고있다.(그림 2-13) Build

페지의 Save all files before build검사칸을 선택하면 프로젝트를 콤파일할 때 JBuilder는 자동적으로 모든 파일을 보관한다.

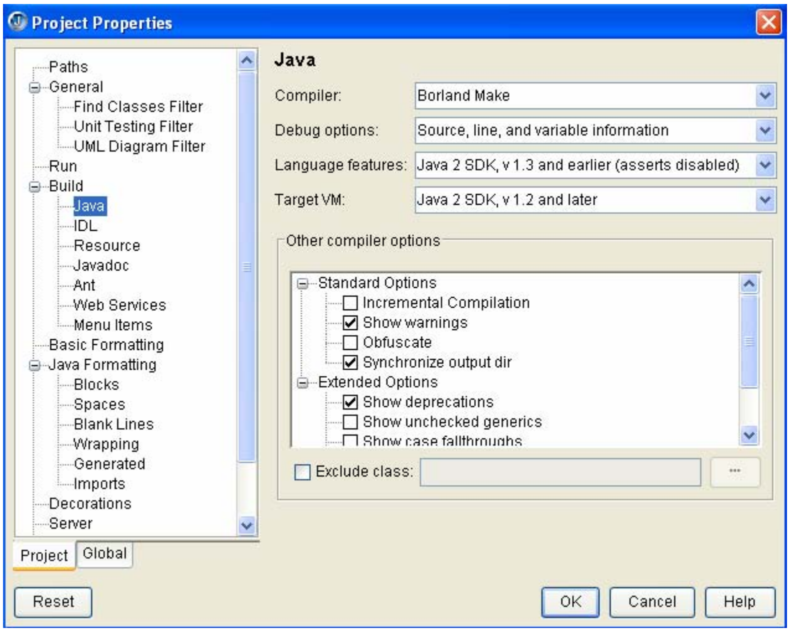


그림 2-13. Project Properties대화칸의 Java페지

Check JSPs for errors at build time검사칸을 선택하면 프로젝트를 콤파일하고 련결할 때 JSP의 모든 오유들을 표시한다. SQLJ Translator마당에서는 None, DB, 또는 Oracle중에서 하나를 선택한다.

Java페지에서는 프로젝트를 콤파일하고 련결시킬 때 체계가 처리하여야 할 조건과 현시할 각종 정보들을 설정한다. 실례로 Debug options복합칸에서 체계의 오유처리설정을 선택하면 JBuilder원천코드에 대한 실행프로그람이 실행중에 있을 때의 각종 변수값정보들을 현시한다. Menu Items페지에서는【Project】차림표에 부분차림표를 추가하고 삭제하는 설정을 진행한다. 실례로 Make와 Rebuild 등을 추가/삭제할수 있다. IDL페지에서는 CORBA를 사용할 때 쓰는 몇가지 관련파라메터들을 설정한다. Resource페지에서는 지원파일이 콤파일되여 련결되였을 때 복사하겠는가 하지 않겠는가를 설정한다.

Java Formatting페지에서는 원천프로그람의 격식을 설정한다. 또한 여기서 preserve Current Line Endings Within Files, Platform Native(\r\n), Windows(\r\n), UNIX(\n)과 Macintosh(\r)인 5개의 단일선택항목중 하나를 선택하여 결속행문자를 지정할수 있다. 그림 2-14에서와 같이 Project Properties대화칸의 Server페지는 하나이상의 응용프로그람봉사기를 선택하는데 쓰인다.【...】단추를 찰칵하여 Edit or Select Server대화칸을 열어 자기의 프로젝트에 요구되는 응용프로그람봉사기를 배비할수 있다.(그림 2-14)

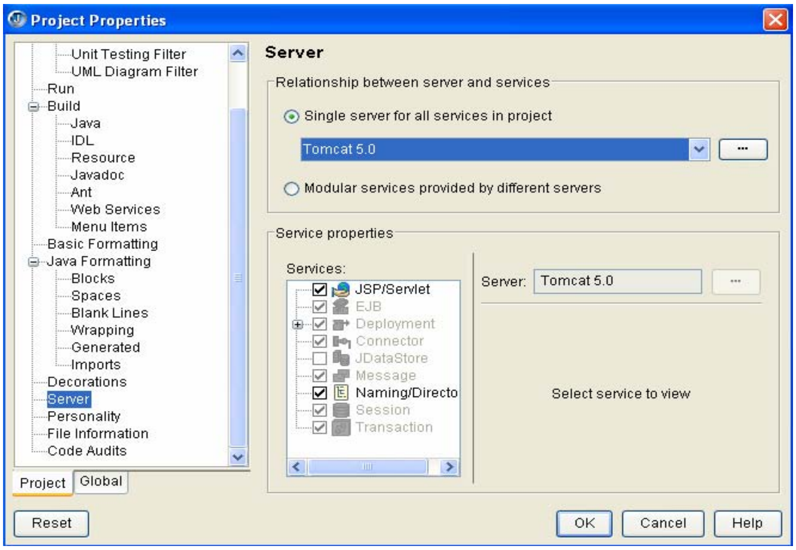


그림 2-14. Project Properties대화칸의 Server페지

2.1.7. 실행차림표 【Run】

【Run】차림표는 여러가지 실행과 오유제거지령들을 제공한다. 여기에는【Run Project】(실행),【Debug Project】(오유제거),【Optimize Project】(최량화),【Configurations】(구성), 【Step Over】(걸음), 【Run to Cursor】(지시자까지 실행), 【Add Watch】(변수관측기능추가) 등이 있다. 이 지령들은 프로그람의 실행과 오유제거를 진행할 때 중요하므로 깊이 학습하여야 한다.

그리고 【Configuration】지령을 찰칵하면 Runtime Configurations대화칸이 열린다. 여기서 Run페지에서와 같이 실행할 주프로그람(여기서는 Java언어의 Application과 Applet프로그람)을 선택할수 있다. 【Step Over】지령과 【Run to Cursor】은 오유제거프로그람과 관련된 지령이다.

2.1.8. 팀차림표 【Team】

기본적으로 팀의 협동개발작업을 진행할 때 리용하는 차림표이다.【Select Project CVS】지령은 주로 현재의 프로젝트에 팀개발판본조종체계를 사용하는데 쓰인다. 이 차림표를 찰칵하면 Select Project VCS대화칸이 나타난다.

2.1.9. 기업차림표 【Enterprise】

이 차림표는 봉사기의 구성과 배비, 기업급설정, EJB프로쎄스와 관련한 지령들을 포함하고있다. 여기에는【Configure Servers】, 【Enterprise Setup】, 【Enterprise Deployment】, 【Create EJB】, 【Client JAR】, 【EJB】, 【Cactus Setup】지령들이 있다.

1. 【Configure Servers】지령

이 지령을 찰칵하면 Configure Servers대화칸이 열리는데 여기서 프로젝트배비에 리용되는 봉사기들에 대한 설정을 진행한다.

2. 【Enterprise Setup】지령

Enterprise Setup대화칸을 현시한다. 여기에 CORBA, Database Drivers, SQL J태브들이 있다. CORBA페지에서는 ORB(VisiBroker, OrbixWeb 등)를 선택하여 JBuilder가 ORB를 사용할수 있도록 하기 위한 추가선택항목들을 설정한다. Database Drivers페지에서는 기존서고에 새로운 파일들을 추가하여 이미 있는 클라스경로(classpath)를 수정한다. SQL J페지는 DB2 또는 Oracle에 대한 SQL J구성을 설정한다.

3. 【Enterprise Deployment】지령

이 차림표지령은 WebLogic Server와 관련한 대화칸을 열고 배비에 관한 설정을 진행한다. 만일 Borland Enterprise Server 5.2.1 혹은 6.0이 우리가 목표하는 응용프로그람봉사기라면 이 차림표지령은 Borland Enterprise Server Deployment조수대화칸을 현시한다. 만일 Borland봉사기가 아니라면 이 지령은 그 봉사기와 관련한 대화칸을 현시한다.

4. 【Create EJB Client JAR】지령

Stubs조수대화칸을 현시하여 JAR가 의뢰기응용프로그람으로 사용되도록 단일한 의뢰기JAR를 생성한다. 그러나 이 항목은 Borland Enterprise Server를 제외한 다른 봉사기들에서는 사용할수 없다.

5. 【EJB】지령

EJB조수대화칸을 현시하여 EJB 1.x Interface Generator, EJB 1.x Bean Generator, Use EJB Test Client들에 대한 설정들을 진행할수 한다.

6. 【Cactus Setup】지령

Cactus Setup조수대화칸을 현시한다. 이것은 Cactus를 가지고 봉사기측단위시험을 하기 위한 프로젝트를 구성하는데 쓰인다.

2.1.10. 도구차림표【Tools】

도구차림표에는 여러가지 환경의 추가선택항목을 제공하고있다. 사용자는 자체로 이 추가선택항목들을 정의할수 있으며 자기의 프로그람작성환경을 설정할수도 있다. 여기에 【Preferences】, 【Configure】(Libraries, JDKs, Obfuscators, File Associations, Palette), 【BeanInsight】, 【JDBC Monitor】, 【Database Pilot】, 【JDataStore Explorer】, 【JDataStore Server】, 【Web Services Explorer】, 【TCP Monitor】, 【Web Services Console】, 【WS-I Basic Profile】, 【Configure External Tools】, 【RMIRegistry】, 【CaliberRM】,

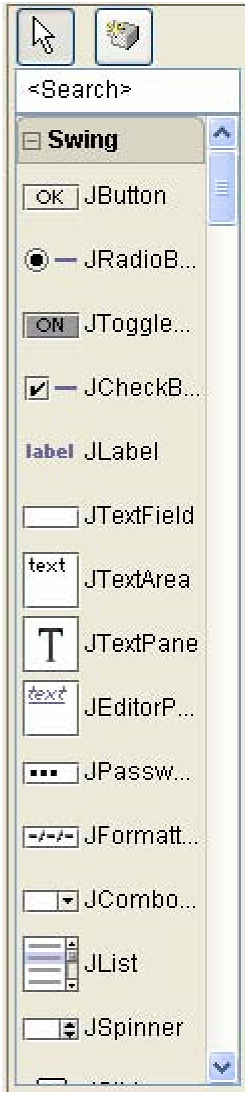


그림 2-15. 부품선택판

【Optimizeit Progress Tracker】, 【Quality Central】들이 있다.

통합개발환경, 편집기 등의 설정을 위한 선택항목인【Preferences】, 클라스서고, JDKs 등과 관련한 구성추가선택항목인【Configure】들이 있으며 자료기지구성과 관련한【JDBC Monitor】, 【Database Pilot】, 【JdataStore Explorer】, 【JdataStore Server】도구들도 있다. 또한 이밖에도 Web봉사, TCT감시, 외부구성도구 등 프로그람개발에 필요한 풍부한 도구들이 이 차림표에 있다.

2.1.11. 창문차림표【Window】

이 차림표는 창문을 관리하는데 쓰이며 【New Browse】를 제외하고는 다른 차림표항목들은 Windows조작체계에서와 사용방법이 같다.

【New Browser】는 기본적으로 다른 GUI(도형사용자대면부)를 창조하고 다른 프로젝트들을 사용하는데 쓰인다. 실례로 2개이상의 프로젝트를 동시에 열려고 하는 경우 지령을 리용하여 2개이상의도형사용자대면부에서 서로 다른 프로젝트들을 열람하고 조작할수 있다.

제2절. 부품선택판

부품선택판은 여러가지 기능을 수행하는 부품들을 기능별로 묶어놓은 서고들이 있는 도구판이다. 부품서고의 매 부품들은 실제상 하나의 객체이며 거기에 기능을 추가하고 수정할수도 있다.

부품선택판은 기능에 따라 16개의 부품서고로 되여있다.(그림 2-15) 아래에서는 제일 많이 사용하는 부품서고들에 대하여 소개한다.

- Swing서고

Swing은 Java기초클라스서고(JFC)의 구성부분으로서 새로운 GUI부품들을 가지고있다. Swing은 100%로 Java로 실현한것이다. 그리고 JDK(Java개발도구묶음), Lightweight UI(User Interface)프레임이 기초로 되고있다. Swing부품은 2개의 묶음 즉 AWT(Abstract Window Toolkit; 추상창문도구묶음)의 순수한 Java판본과 고급한 부품모임(례하면 계층구조보기, 목록 및 태브 등)을 가지고있다.

- Swing Containers서고

Swing Containers에는 일부 Swing부품들이 사용하는 용기들이 포함되여있다.

- DataExpress서고

Borland회사는 자체의 자료기지처리부품들을 가지고 있는데 이 부품들이 DataExpress서고에 속해있다.

- AWT서고

AWT는 Java기초클라스서고(JFC)의 하나로서 아래와 같은 기능들을 가지고있다.

· 풍부한 사용자대면부품
· 강력한 사건처리모형
· 도형 및 화상도구(형태, 색갈, 서체클라스들을 포함)
· 배치관리기
  능동적인 창문배치를 할수 있다.

기존가동기반의 오려둠판으로서 오려내기와 붙이기를 할수 있다.

그밖에 XML, EJB, InternetBeans, CORBA 서고 등 JBuilder에서 사용하는 고급한 프로그람작성기능부품들을 포함하고있는 서고들이 있다.

아래에서 이 부품들을 리용하는 프로그람을 작성해보도록 하자.

2.2.1. Project Wizard를 사용한 프로젝트의 창조

JBuilder의 Project Wizard를 사용하여 이름이 usingSwing.jpx인 프로젝트를 만들어 보자. 그 단계는 아래와 같다.

단계

1 【File】의 【New Project】지령을 선택하면 Project Wizard대화칸이 나타난다.

2 【Name:】본문칸에서 《untitled》를 《usingSwing》으로 고치고 다른 추가선택항목들은 기정으로 한다.

3 【Next】단추를 찰칵하여 다음 페지로 들어간다.

4 이 페지에서는 기정으로 설정하고 【Next】단추를 찰칵하여 마지막 페지에로 들어간다. 【Title:】마당에 《usingSwing》을 입력한다. 【Finish】단추를 찰칵하면 프로젝트작성이 끝나며 JBuilder는 자동적으로 프로젝트 usingSwing을 생성한다.

2.2.2. Application Wizard를 사용한 Java응용프로그람의 창조

계속하여 Application Wizard를 사용하여 Java응용프로그람을 만들어 보자. 구체적인 단계는 다음과 같다.

단계

1 【File】차림표의 【New】지령을 선택하면 New대화칸이 나타난다.

2 여기서 Application아이콘을 찰칵하여 Application Wizard대화칸을 연다. 이 대화칸에서 추가선택항목들을 기정으로 설정한다. 【Next】단추를 찰칵하여 다음 페지로 들어간다.

3【Class:】본문칸에 《usingSwingFrame》을 입력하고【Title:】본문칸에 《Swing 부품을 사용한 Java응용프로그람》을 입력한다.

4 【Next】단추를 찰칵하여 다음 페지로 넘어간다.

5 이 페지에서는 추가선택항목들을 기정으로 설정한다. 【Finish】단추를 찰칵하면 체계는 자동적으로 Application1.java와 usingSwingFrame.java인 2개의 프로그람을 작성하여 프로젝트에 추가한다. 또한 응용프로그람대면부와 그에 대응하는 원천프로그람들이 자동적으로 생성된다.

2.2.3. 기본창문대면부설정

그림 2-16과 같이 창문우에 부품을 설치한다. 구체적인 단계는 다음과 같다.

단계

1 부품선택판에서 Swing Containers서고의 javax.swing.JTabbedPane부품을 선택한다.

2 Design방식에서 이 단추를 찰칵하면 jTabbedPane1부품객체가 추가된다.

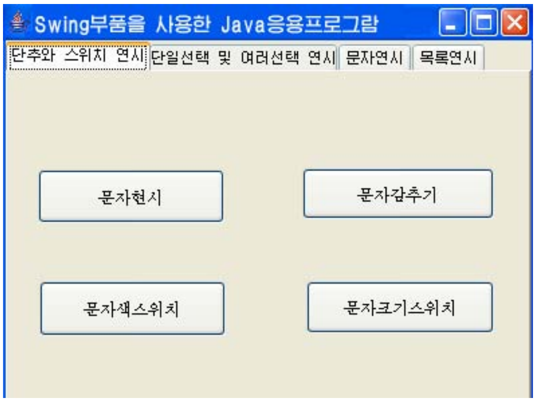
3 SwingContainers서고의 javax.swing.JPanel를 선택한다.

4 jTabbedPane1부품객체에 4개의 JPanel부품(jPanel1, jPanel2, jPanel3, jPanel4)을 추가한다.

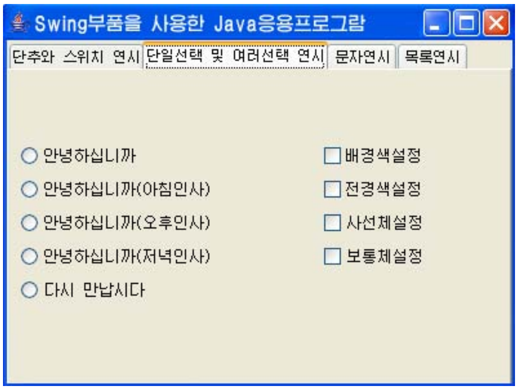
5 Inspector에서 속성태브를 찰칵하고 jPanel1, jPanel2, jPanel3, jPanel4의 constraints속성들을 각각 《단추와 스위치연시》, 《단일선택 및 여러선택 연시》, 《문자연시》와 《목록연시》로 수정한다.

6 Inspector에서 jPanel1, jPanel2, jPanel3, jPanel4의 layout속성들을 각각 《XYLayout》, 《BorderLayout》, 《BorderLayout》, 《XYLayout》로 수정한다.

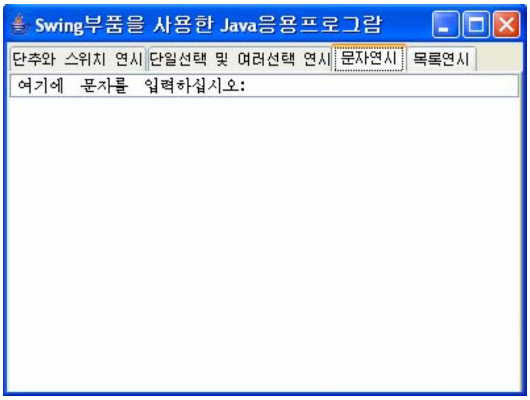
7 그림 2-16의 ㄱ)와 같이 jPanel1에 표 2-10의 부품들을 추가한다.



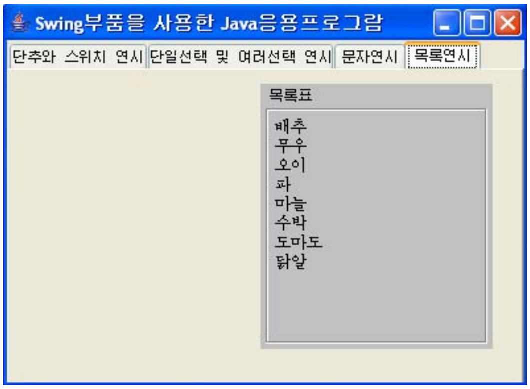
ㄱ)



ㄴ)



ㄷ)



ㄹ)

그림 2-16. 주창문대면

표 2-10. jPanel1 의 부품객체

부 품 이 름	설 명
jLabel1	text 속성의 기정값을 지운다.
jButton1	text 속성을 《문자현시》로 설정한다.
jButton2	text 속성을 《문자감추기》로 설정한다.
jToggleButton1	text 속성을 《문자색스위치》로 설정한다.
jToggleButton2	text 속성을 《문자크기스위치》로 설정한다.

8 jPanel2에 부품을 추가하기 전에 먼저 대면부를 분할한 다음 그림 2-16의 ㄴ)와 같

이 jPanel2에 부품을 추가한다.

9 우선 jPanel5와 jPanel6을 추가하고 jPanel5의 constraints속성을 《center》로, layout속성을《BoxLayout2》로 설정한다. 그리고 jPanel6의 constraints속성을《south》로, layout속성을 《BorderLayout》로 설정한다.

10 jPanel6에 하나의 jLabel2부품을 추가하고 text속성의 기정값을 지운다.

11 jPanel5에 jPanel7과 jPanel8인 2개 부품객체를 추가한다. 그리고 2개의 부품들의 layout속성을 《VerticalFlowLayout》로 설정한다. 이렇게 하면 여기에 단일선택과 여러선택 단추객체들을 추가할수 있다.

12 jPanel7에 5개의 javax.swing.JRadioButton부품을 추가하고 text속성을 《안녕하십니까》, 《안녕하십니까(아침인사)》, 《안녕하십니까(오후인사)》, 《안녕하십니까(저녁인사)》, 《다시 만납시다》로 각각 설정한다.

14 Swing서고의 ButtonGroup1부품객체를 추가한다.

15 5개 javax.swing.JRadioButton부품의 buttonGroup속성을 모두 《ButtonGroup1》로 수정한다.

16 jPanel8에 4개의 javax.swing.JCheckBox부품을 추가하고 text속성을 각각《배경색설정》, 《전경색설정》, 《사선체설정》, 《보통체설정》으로 한다.

17 그림 2-16의 ㄷ)와 같이 jPanel3에 표 2-11과 같은 부품들을 추가한다.

18 그림 2-16의 ㄹ)와 같이 jPanel4에 표 2-12와 같은 부품들을 추가한다.

표 2-11. jPanel3 에 추가할 부품

부 품 이 름	설 명
jTextField1	text 속성을 《여기에 문자를 입력하십시오:》로, constraints 속성을 《North》로 설정한다.
jTextArea1	text 속성의 기정값을 지운다. constraints 속성을 《center》로, enabled 속성을 《false》로 설정한다.

표 2-12. jPanel4 에 추가할 부품

부 품 이 름	설 명
jLabel3	text속성의 기정값을 지운다.
jList1	selectionMode속성을《MULTIPLE_INTERVAL_SELECTION》으로 설정한다. background속성을 《lightGray》로, border속성을 《titledBorder2》로, 그리고 border속성의 오른쪽단추를 찰칵하여 《title》속성마당에 《목록표》를 입력한다.

19 Source태브를 찰칵하여 코드편집창으로 절환하고 jList1을 선언한 위치를 찾아 다음의 코드를 추가한다.

Private String【 】 data={“배추”, “무우”, “오이”,“파”, “마늘”, “수박”,“도마도”, “닭알”};

jList1의 선언부를 《private JList jList1=new JList(data);》로 수정한다.

이렇게 하면 대면부를 기본적으로 완성하였다. 아래에서는 실제적인 코드를 작성한다. 즉 매개 부품에 대응하는 코드들을 추가한다.

2.2.4. 코드작성

1. jButton1과 jButton2에 대한 코드작성

우선 UI내용판이나 구조판의 부품계층구조에서 【Jbutton1】단추를 찰칵한다. 그리고 Inspector에서 action Performed사건을 마우스로 두번 찰칵하면 코드편집창으로 넘어간다. 이때 jButton1_action Performed메쏘드가 창조된다. 여기에 《jLabel1.setText(“나의 문자연시”); 》를 입력하면 【jButton1】단추에 대한 코드가 완성된다. 같은 방법으로 【jButton2】단추에 《jLabel1.setText(“나의 문자감추기”);》코드를 입력한다.

2. jToggleButton1과 jToggleButton2에 대한 코드작성

우선 UI내용판이나 구조판의 부품계층구조에서 【jToggleButton1】단추를 찰칵하고 Inspector의 사건태브를 찰칵한다. 다음 stateChanged사건을 마우스로 두번 찰칵하면 코드편집창으로 넘어가며 jToggleButton1_stateChanged메쏘드가 자동적으로 창조된다.

이 메쏘드에 입력하여야 할 프로그람은 다음과 같다.

```
if(jToggleButton1.isSelected())
jLabel1.setForeground(Color.blue);// 선택시 문자색은 푸른색
else
jLabel1.setForeground(Color.red);//그렇지 않으면 붉은색
```

이렇게 하여 【jToggleButton1】단추에 대한 코드를 완성한다.

이런 방법으로 【jToggleButton2】단추에 다음과 같은 코드를 입력한다.

```
Font font1=new Font(“plain”,Font.PLAIN,16);
Font font2=new Font(“plain”,Font.PLAIN,24);
if(jToggleButton2.isSelected())
jLabel1.setFont(font2);
else
jLabel1.setFont(font1);
```

3. jRadioButton1, jRadioButton2, jRadioButton3, jRadioButton4, jRadioButton5에 대한 코드작성

우선 UI내용판이나 구조판의 부품계층구조에서【JRadioButton1】단추를 찰칵한다. 계속하여 Inspector의 사건태브를 찰칵한다. 거기서 actionPerformed사건을 마우스로 두번 찰칵한다. 이때 Soruce페지에 들어가며 동시에 jRadioButton1_actionPerformed메쏘드가 창조된다. 여기에《jLabel2.setText("안녕하십니까");》를 입력하여【jRadioButton1】단추에 대한 코드를 완성한다.

같은 방법으로 【jRadioButton2】단추에 《jLabel2.setText("안녕하십니까(아침인사)");》코드를 입력한다.

【jRadio Button3】단추에 《jLabel2.setText("안녕하십니까(오후인사)");》코드를 입력한다.

【jRadio Button4】단추에 《jLabel2.setText("안녕하십니까(저녁인사)");》코드를 입력한다.

【jRadio Button5】단추에 《jLabel2.setText("다시 만납시다");》코드를 입력한다.

4. jCheckBox1, jCheckBox2. jCheckBox3과 jCheckBox4에 대한 코드작성

우선 UI내용판이나 구조판의 부품계층구조에서 【jCheckBox1】단추를 찰칵하고 계속하여 Inspector의 사건태브를 찰칵한다. 다음 stateChanged사건을 마우스로 두번 찰칵한다. 그러면 코드편집창에 jCheckBox1_stateChanged메쏘드가 창조된다. 이 메쏘드에 대응하는 코드는 다음과 같다.

```
If(jCheckBox1.isSelected())
jPanel6.setBackground(Color.blue);
else
    jPanel6.setBackground(Color.lightGray);
```

이렇게 하여 jCheckBox1단추에 대한 코드를 완성한다.

같은 방법으로 【jCheckBox2】단추에 대한 코드를 다음과 같이 입력한다.

```
if(jCheckBox2.isSelected())
jLabel2.setForeground(Color.red);
else
jLabel2.setForeground(Color.black);
```

같은 방법으로 jCheckBox3단추에 대한 코드를 다음과 같이 입력한다.

```
Font font1=new Font("plain",Font.PLAIN,12);
Font font2=new Font("plain",Font.PLAIN,24);
Font font3=new Font("italic",Font.ITALIC,12);
Font font4=new Font("italic",Font.ITALIC,24);
if(jCheckBox4.isSelected()&&jCheckBox3.isSelected())
  {  jLabel2.setFont(font4);}
  else if(jCheckBox4.isSelected()&&!jCheckBox3.isSelected())
{
    jLabel2.setFont(font2);
}
else if(!jCheckBox3.isSelected()&&!jCheckBox4.isSelected())
{
    jLabel2.setFont(font1);
}
else if(jCheckBox3.isSelected()&&!jCheckBox4.isSelected())
{
    Label2.setFont(font3);
}
}
```

마찬가지로 jCheckBox4단추에도 코드를 아래와 같이 입력한다.

```
Font font1=new Font("plain",Font.PLAIN,12);
Font font2=new Font("plain",Font.PLAIN,24);
Font font3=new Font("italic",Font.ITALIC,12);
Font font4=new Font("italic",Font.ITALIC,24);
if(jCheckBox4.isSelected()&&jCheckBox3.isSelected())
{   jLabel2.setFont(font4);}
else if(jCheckBox4.isSelected()&&!jCheckBox3.isSelected())
{
    jLabel2.setFont(font2);
}
else if(!jCheckBox3.isSelected()&&!jCheckBox4.isSelected())
{
```

```
      jLabel2.setFont(font1);
   }
 else if(jCheckBox3.isSelected()&&!jCheckBox4.isSelected())
    {
        jLabel2.setFont(font3);
      }
 }
```

5. jTextField1에 대한 코드작성

우선 UI내용판이나 구조판의 부품계층구조에서 【jTextField1】단추를 찰칵한다. 다음 actionPerformed사건을 마우스로 두번 찰칵한다. 이때 코드편집창이 열리고 jTextField1_actionPerformed메쏘드가 창조된다.

코드는 다음과 같다.

```
jTextAreal.append (jTextField1.getText()+ "\n" );
jTextField1.setText( " " );
```

이렇게 하여 【jTextField1】단추에 대한 코드작성작업이 끝난다.

6. jList1에 대한 코드작성

우선 UI내용판이나 구조판의 부품계층구조에서 【jList1】단추를 찰칵한다. 그리고 Inspector의 사건태브를 찰칵한 다음 valueChanged사건을 마우스로 두번 찰칵한다. 이때 코드편집창이 현시되며 동시에 jList1_valueChanged메쏘드가 창조된다.

《jLabel3.setText((String)jList1.getSelectedValue());》을 입력하여 【jList1】단추에 대한 코드입력을 완성한다.

2.2.5. 프로그람의 콤파일과 실행

프로그람코드작성이 끝난후 프로그람을 콤파일한다. 그다음 련결시키고 실행하면 된다. 【Run】차림표의 【Run Project】지령을 선택하면 모든 작업을 완성할수 있다. 프로그람 실행결과는 그림 2-17에서 보여준다.

이 실례를 통하여 각종 부품객체들에는 JPanel, JTabbedPane, JButton, JLabel, JToggleButton, JCheckBox, JRadioButton, ButtonGroup, JTextArea, JList 등이 있다는것을 알수 있다. 이 부품들은 비교적 자주 쓰는것들인데 구체적인 사용방법은 JDK 클라스서고를 참고하면 된다.

Java로 작성한 응용프로그람들은 여러 가동기반들에서 배비할수 있다.

전통적인 UI설계기술에서는 여러가동기반에서 외관상 일치하지 않기때문에 반드시 부품의 절대위치와 척도를 지정하여야 한다.

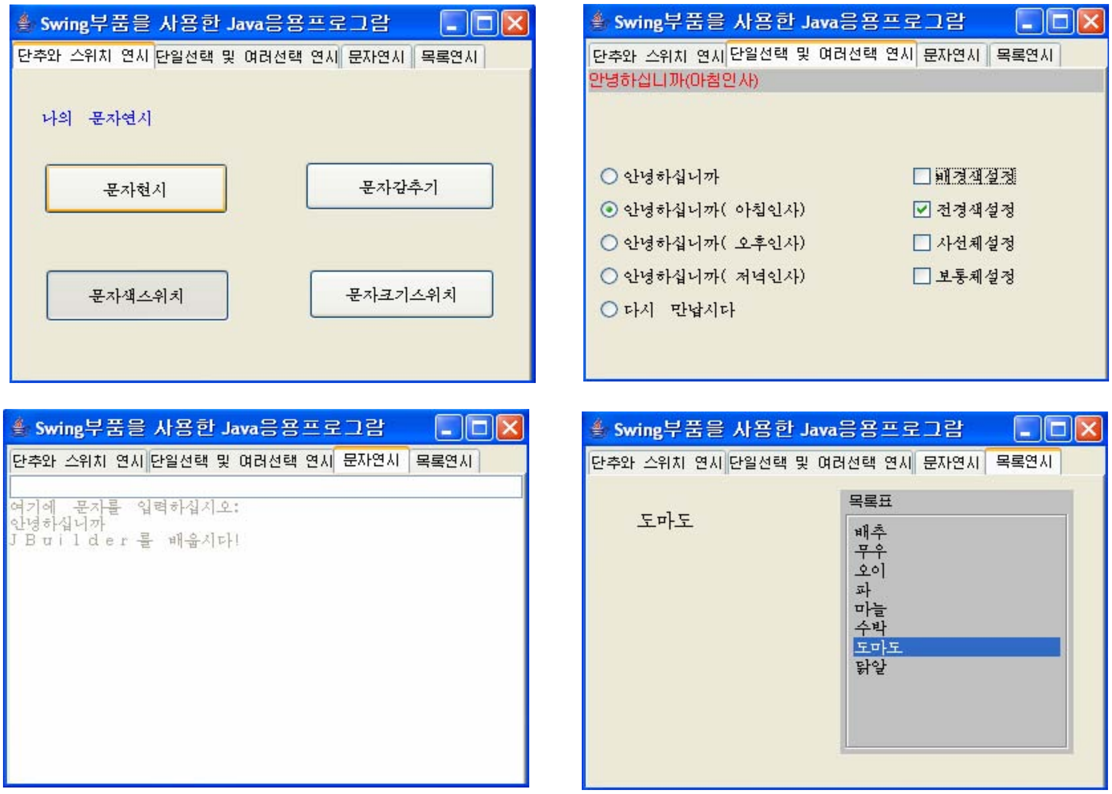


그림 2-17. 프로그람실행 결과

이 문제를 해결하기 위하여 Java에서는 이식가능한 배치관리기체계를 제공하고있다. 배치관리기는 부품의 위치를 정확히 확정할수 있으며 어떤 형태의 서체, 화면분해률과 가동기반을 사용하는가에 무관계하다. 매 부품에 대하여 JBuilder는 Java에서와 같이 다음의 배치관리기를 제공하고있다. 즉 BorderLayout, FlowLayout, GridLayout, CardLayout, GridBagLayout, Null, XYLayout, PanelLayout, VerticalFlowLayout, BoxLayout2, OverLayout2 등이 있다. 이 배치관리기들에 정통하면 대면부설계의 효과성과 속도를 높일수 있다.

2.2.6. JavaBeans부품을 부품선택판에 추가

부품선택판에 JavaBeans부품을 추가할수 있다. 우선 부품선택판의 임의의 곳에서 마우스오른쪽단추를 찰칵하여 지름차림표를 열고 【Properties】항목을 찰칵한다. 이때 현시

되는 【Palette Properties】대화칸에서 【Add】단추를 사용하면 부품을 부품선택판에 추가할수 있고 【Delete】단추를 사용하면 부품선택판에서 삭제할수 있다. 또한 부품들의 이름도 바꿀수도 있으며 부품선택판에서의 부품현시순서를 변경할수도 있다.(그림 2-18)

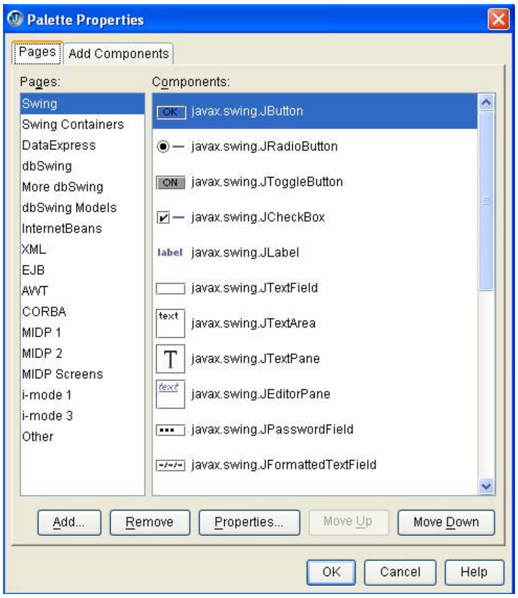


그림 2-18. Palette Properties대화칸

주의 여기에서의 삭제는 JBuilder로부터 이 부품을 완전히 삭제한다는것을 의미하지 않으며 단지 부품선택판에 현시되지 않는다. 만일 사용자가 요구하면 이 부품을 다시 부품선택판에 추가할수 있다.

만일 부품선택판에 부품을 추가하려면 우선 Library를 추가하여야 한다. 차림표【Tools】의【Configure】/【Libraries...】지령을 선택하면 그림 2-19와 같이 Configure Libraries 대화칸을 열수 있다.

우선 【Add folder】단추를 찰칵하여 자기의 클라스서고 파일서류철을 창조한다.(여기서는 User Home이다.) 그 다음 【New...】단추를 찰칵하면 이때 New Library Wizard 대화칸이 나타난다. 자기가 클라스서고에 창조하려는 이름을 입력하고(여기에서는 mapxtreme) 방금 창조한 클라스서고 파일서류철을 선택한다. 그리고 다시 【Add...】단추를 찰칵하여 자기클라스서고의 파일경로와 .jar서고파일을 추가한다. 다음에 【Ok】단추를 찰칵하여 새 클라스서고파일의 추가작업을 완성한다.

마지막 결과는 그림 2-19와 같다. 마찬가지로 다른 클라스서고파일을 추가할수도 있다.【OK】단추를 찰칵하여 모든 작업을 완성하고 조작결과를 보관한다.

아래에서 부품선택판에 부품을 추가하는 방법에 대하여 고찰한다. 차림표 【Tools】의 【configure】/【palette】지령을 선택하면 그림 2-18과 같은 부품선택판의 속성대화칸을 열수 있다.

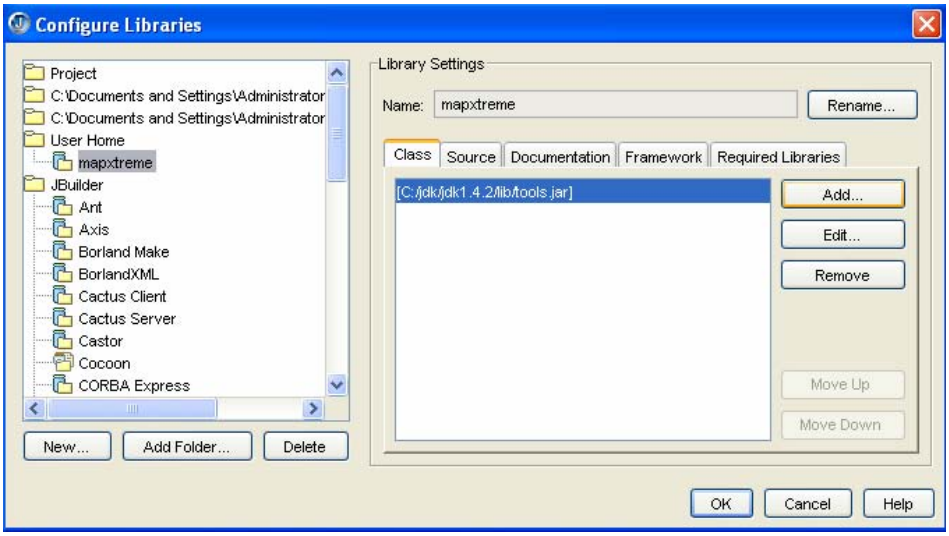


그림 2-19. Configure Libraries대화칸

우선 【Add】단추를 찰칵하여 MyBeans라는 새로운 부품페지를 추가한다.(여기서 자신의 부품페지이름을 추가할수 있다.) 다음에 Add components태브를 찰칵하여 그림 2-20과 같은 페지에로 들어간다. 【select Library...】단추를 찰칵하면 서고를 선택하는 대화칸이 나타난다. 여기서 방금 우리가 본 Ant가 있는데 그것을 지정하고 다시 【OK】단추를 찰칵하여 클라스서고를 선택한다.



Library대화칸에는 【New】단추가 있다. 이 단추를 찰칵하여도 New Library Wizard조수대화칸에 들어갈수 있다

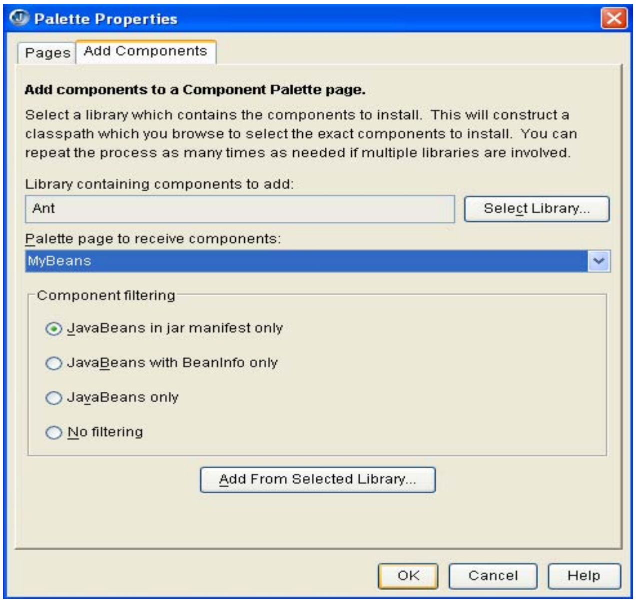


그림 2-20. 부품속성대화칸의 Add Components페지

Palette page to receive components의 내리펼침칸에서 방금 창조한 MyBeans부품페지를 선택하고 다음에 【Add From Selected Library...】단추를 찰칵하면 자기의 클라스의 JavaBeans부품이 부품페지에 추가되는것을 볼수 있다.

제3절. 구조판

구조판(Structure)은 통합개발환경의 왼쪽 아래에 있다. 이것은 프로그람개발에서 중요한 역할을 한다. 구조판의 구체적인 역할은 사용자에게 객체의 공개선언부의 접근속성을 제공하는것이다.(례하면 읽어들이기와 수정 등) 동시에 구조판은 객체의 사건(Events)에 대한 설정을 제공해준다. 여기에서 객체지향기술에서 요구하는 많은 과제들을 완성할수 있으며 객체의 속성과 사건의 변경은 바로 프로그람작성의 전부라고 볼수 있다.

JBuilder에서 객체의 속성은 일반적으로 4가지가 있다.

- Text속성

Text속성은 속성값을 직접 입력할수 있다.

- Enumerated속성

구조판은 내리펼침목록을 현시하여 사용자가 목록에서 속성값을 선택하게 한다.

- Set속성

set에는 일련의 속성들이 포함된다. 사용자는 이 속성들에 대한 선택을 통하여true/false 속성을 선택한다.

- Object속성

그 자체가 하나의 객체속성이며 확장할수 있는 속성목록으로 된다. 이 항목에 대하여 사용자는 단독편집할수 있다.

현재 매개 속성의 구체적인 의미를 리해하지 못할수 있다. 그러나 뒤부분에서 제공하는 속성들에 대한 구체적인 사용방법들을 통하여 속성의 의미를 깊이 리해할수 있다.

제4절. UI설계기

시각적인 프로그람작성에서 사람과 콤퓨터사이의 대화는 기본적으로 일부 창문들과 대화칸들로 실현한다. JBuilder는 이 창문들과 대화칸들에 대한 프레임을 실현해준다. 부품들은 프레임상에 놓이며 이 부품들을 가지고 복잡한 기능들을 실현할수 있다.

UI설계기는 사용자프로그람의 작성과정에서 가장 많이 사용하는 도구이다. 사용자는 UI설계를 하는 경우 파일보기태브의 Design을 찰칵하기만 하면 UI설계를 할수 있다.

앞에서 든 실례들에서 알수 있는바와 같이 UI설계기에 JBuilder가 제공하는 많은 부품들을 추가할수 있으며 프레임에서 이 부품들을 이동시키거나 크기를 변경할수 있다는것을 보았다. 이것이 바로 시각화프로그람설계이다. 사용자는 프레임안에서 작업하게 되며 이것은 마치 생산직장안에서 기계를 조립하는것과 같다. 또한 JBuilder는 자동적으로 이 대면부에 대해 간단한 코드를 생성하며 사용자에게 편리한 프로그람작성환경을 제공한다. 그다음 사용자는 속성을 설정하고 구체적인 기능을 실현하는 코드를 작성하게 된다.

제5절. JBuilder편집기

Source태브를 찰칵하면 원천코드편집방식으로 들어간다. 이때 JBuilder는 자동적으로 이와 관련한 대부분의 코드를 생성해준다. 그러므로 사용자는 오직 JBuilder에서 이미 생성한 코드에 자기가 요구하는 업무론리를 실현하는 코드를 더 추가하면 된다. 실례로 어떤 프로그람에 단추를 찰칵하면 실행을 끝마치는 사건에 대한 응답코드를 추가하는 경우 이미 생성한 코드뒤에 간단히 명령문 《System.exit(0);》을 추가하면 된다. 대응하는 코드는 다음과 같다.

```
void jButton8_actionPerformed(ActionEvent e){
System.exit(0);
}
```

원천코드편집기는 사용자의 요구에 따라 형식을 바꿀수 있다. 이 설정은 【Tools】→【Preferences...】지령을 선택하면 나타나는 【Preferences】대화칸에서 진행한다. 실례로【Editor】항목을 찰칵하면 나타나는 대화칸에서 편집기의 색갈, 코드미리보기, 형판, Java 구조 등에 대한 설정들을 진행할수 있다.

원천코드편집기에서 도움말을 리용하려면 유표를 검색하려는 열쇠어 혹은 객체 등에 놓고 【F1】지름건을 누르면 그에 해당한 도움말이 나타난다.

콤파일하는 경우 오유가 생기면 통보문판에 오유정보를 현시하고 원천코드편집기에서 오유가 발생한 행을 색갈로 표시한다.

원천코드편집기에서 다른 파일에 접근하려면 프로젝트판에서 그 파일을 선택하면 된다. Source태브가 선택된 상태에서 구조판은 내용판에 있는 현재 파일의 구조를 현시한다. Java 파일인 경우 이 창문에는 계층구조형식으로 모든 메쏘드, 속성과 사건들이 현시된다. 구조판의 클라스나 대면을 두번 찰칵하면 그것의 상위클라스들을 찾을수 있다. 그리고 코드편집기에서 찾으려는 메쏘드, 속성과 사건을 고속으로 찾아볼수 있다. 이밖에도 코드편집기에서 수정한 내용은 동시에 사용자대면부에 있는 아이콘들에 반영된다.

JBuilder편집기는 본문탐색, 본문강조현시, 코드형판과 코드미리보기 기능들을 지원한다. JBuilder의 코드미리보기기능(CodeInsight)은 튀여나옴창문을 현시하여 명령문과 관련이 있는 정보들을 현시해줌으로써 Java코드작성을 방조해준다.(그림 2-21)

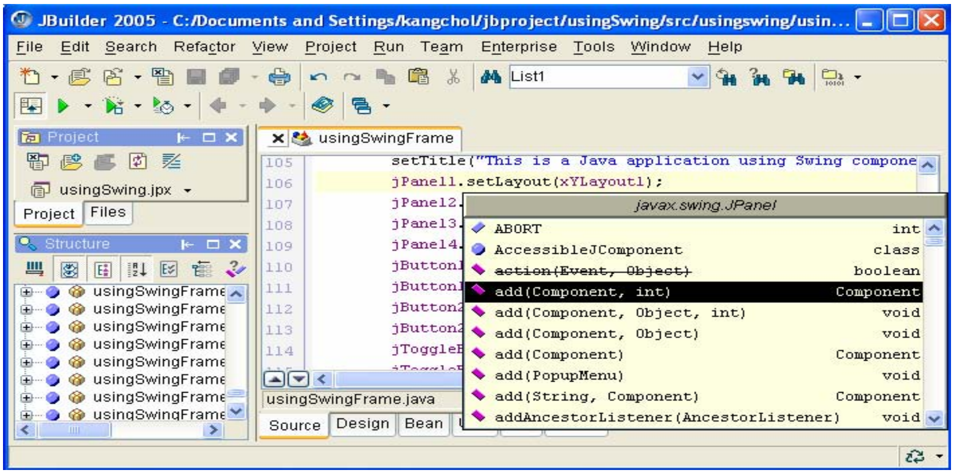


그림 2-21. JBuilder의 코드미리보기기능

즉 코드미리보기기능은 현재 선택한 코드부분과 련관있는 자료성원과 메쏘드목록(MemberInsight), 현재 코드에서 사용하려는 메쏘드의 파라메터목록(ParameterInsight), 현재 경로를 통하여 접근할수 있는 클라스목록(ClassInsight), 오유(ErrorInsight)들을 현시해준다. 오유들이 제거되면 변수값과 표현식의 값을 현시한다.

코드형판의 사용역시 프로그람의 작성속도를 가속시켜 준다. JBuilder는 이미 일련의 형판들을 가지고있다. 여기에는 클라스선언부, if문, if else문, try/catch문 및 while문 등

의 형판들이 포함되여있다. 편집기에서 코드형판의 이름을 입력하면 편집기는 자동적으로 이 형판의 코드를 현시한다. 또한 지름건【Ctrl+J】을 리용하여 코드형판을 그림 2-22와 같이 능동으로 할수 있다.

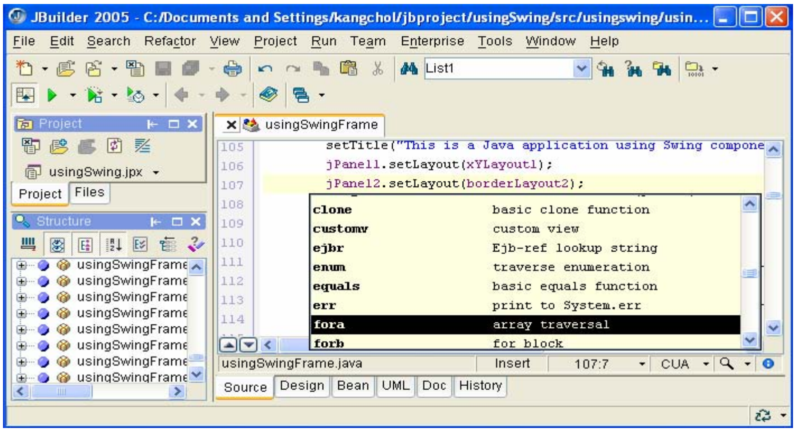


그림 2-22. JBuilder의 코드형판기능

한편 【Tools】→【Preferences...】를 선택하여 【Editor】의 CodeInsight페지에서는 코드미리보기에 관한 추가설정을 할수 있으며 Templates페지에서는 코드형판을 편집, 추가, 삭제를 할수 있다.(그림 2-23)

여기서 【Add】단추를 리용하면 새로 설정하려는 코드형판이름과 설명을 추가할수 있으며 【Edit】단추를 리용하면 설정한 코드형판의 이름과 설명을 편집할수 있다. 임으의 코드형판의 코드내용을 편집하려는 경우에는 우선 이 형판이름을 선택한 다음 [Code:]에서 수정하거나 추가할수 있다. 【Reset】단추를 찰칵하면 체계가 설정한 기정상태로 회복된다.

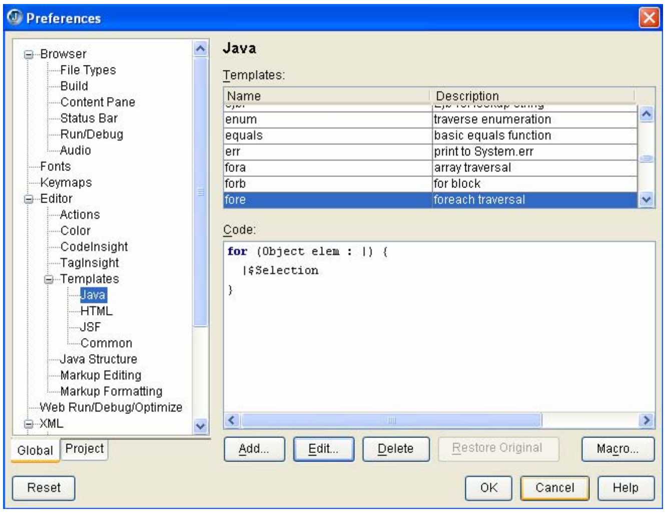


그림 2-23. 코드형판대화칸의 설정

제6절. JBuilder의 오유검출과 오유제거기술

이 절에서는 JBuilder에서의 콤파일오유검출과 오유제거기능을 설명한다. JBuilder편집기는 자체로 오유를 검출하는 기능을 가지고있다. 실례로 jToggleButton2을 jToggleButton으로 수정하면 그림 2-24와 같은 결과가 나타난다.

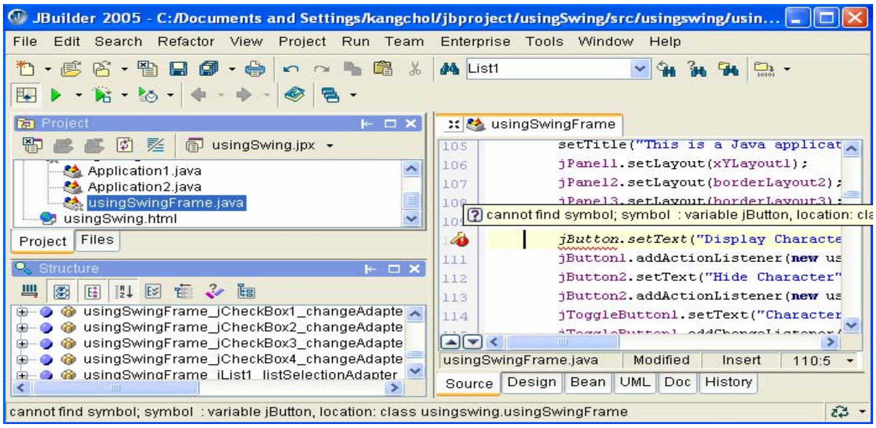


그림 2-24. 편집기의 오유검출

그림 2-24에서 밑선과 오유를 표시하는 아이콘을 통해 오유를 나타내고있다. JBuilder편집기는 기본적으로 문법상의 오유를 제거하고있지만 제거할수 없는 오유에 대하여서는 우와 같이 오유위치와 류형을 밝혀준다.

편집기에서는 또한 임의의 명령문의 앞을 마우스로 찰칵하면 이 행을 자동적으로 중단점으로 설정한다.(그림 2-25) 우에서 본 중단점은 단지 행중단점으로서 이런 중단점에는 그림 2-26와 같이 모두 5개의 류형이 있다.

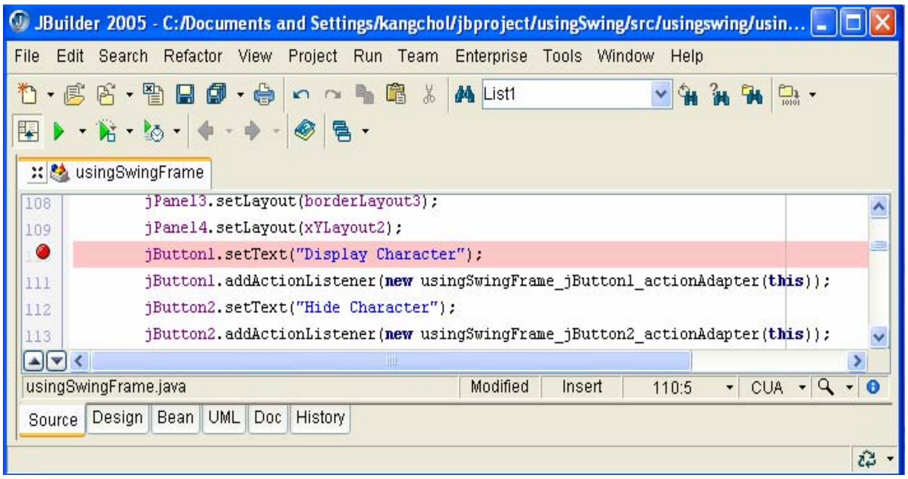


그림 2-25. 중단점설정

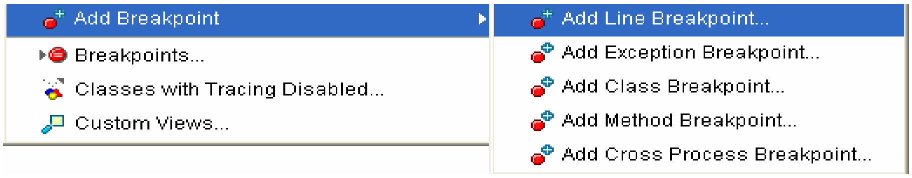


그림 2-26. 중단점의 설정류형

표 2-13에서는 5개 류형의 중단점에 대하여 설명하고있다.

표 2-13. 중단점의 종류

중단점류형	설　　명
line 중단점	원천코드의 구체적인 행에 설정한다. 오유제거기는 그 행에서 중지된다.
exception 중단점	구체적인 례외가 버려질 때 오유제거기가 중지된다.
class 중단점	구체적인 클라스의 임의의 메쏘드가 호출되거나 구체적인 클라스가 초기화될 때 오유제거기가 중지된다.
method 중단점	구체적인 클라스의 구체적인 메쏘드가 호출될 때 오유제거기가 중지된다.
cross-process 중단점	구체적인 처리에서 구체적인 클라스의 임의의 메쏘드나 어떤 구체적인 메쏘드가 단계별로 실행될 때 오유제거기가 중지된다.

【Run】차림표의 【Debug Project】지령을 선택하면 프로그람의 오유제거대면부에 들어간다. 이때 JBuilder는 그림 2-27와 같은 오유제거정보현시창문을 펼친다.

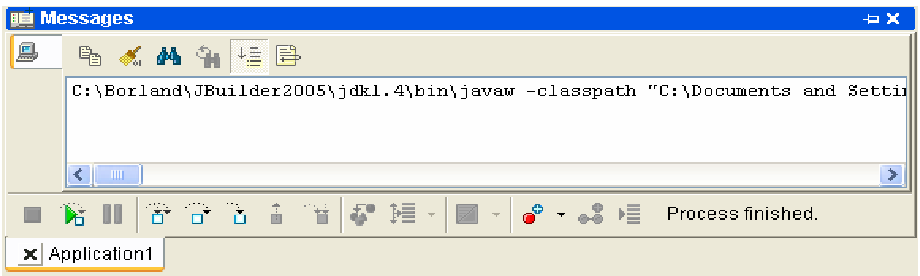


그림 2-27. 오유제거정보현시창문

Console output, input, errors창문에서 입력, 출력, 각종 오유들을 현시할수 있다. 프로그람의 오유제거와 실행시에 체계는 례외오유도 현시할수 있다. 그림 2-28과 같이 프로그람실행중의 례외오유를 창문에 현시하고있다. 통보문판에서는 오유가 나온 파일이름과 프로그람행을 현시하고있다. 이 부분을 찰칵하면 편집기에서 이 파일에 들어가 진한 색갈로 오유가 생긴 프로그람행을 현시하는데 그림 2-28에서는 usingSwingFrame.java파일의 205행에서 오유가 발생하였다고 오유통보문을 현시하고있다. 이때 이 부분의 코드를 자세히 조사해보아야 한다.

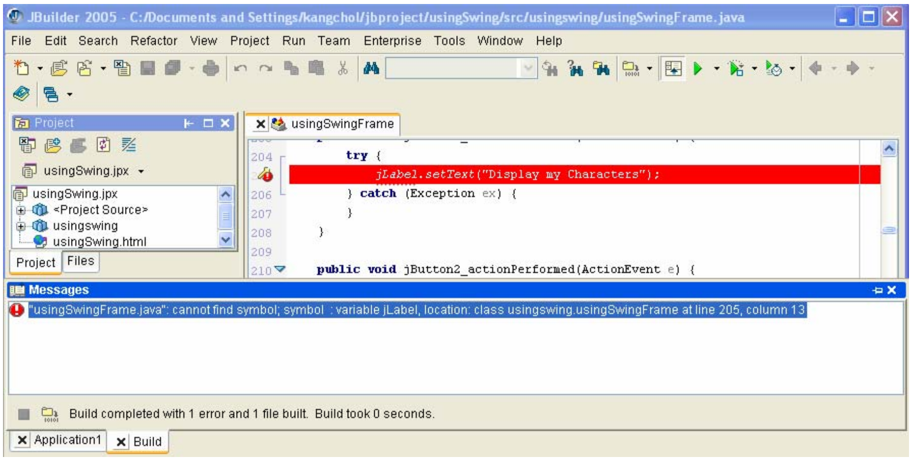


그림 2-28. 례외오유의 조사

오유제거기가 없는 경우 실행시의 오유를 알아내기가 매우 어렵다. 그것은 콤파일기가 오유에 대한 아무런 상황도 알려주지 않기때문이다. 이때 보통 오유를 알아낼수 있는 유일한 방도는 프로그람실행결과뿐이다. 실례로 화면에서의 비정상적인 결과현시와 실행시의 오유발생통보 등을 통하여 그 원인을 알아볼수 있다.

반면에 오유제거기가 있는 경우는 오유류형을 빨리 알아낼수 있다. 오유제거기를 사용하면 구체적인 프로그람위치까지 실행할수 있으므로 프로그람을 한걸음 한걸음 실행하여 매걸음에서의 상태를 관찰할수 있다. 그리고 원천코드를 수정하고 프로그람을 다시 콤파일하여 계속 검사해 나갈수 있다. 만일 프로그람이 실행시 오유를 버리기하였다면 Console output, input, errors창문에서 탄창궤적(stack trace)을 출구한다. 이렇게 하여 오유를 편리하게 빨리 찾아보아 프로그람의 오유를 제거할수 있다.

그림 2-29와 같은 창문은 스레드를 호출할 때 사용하는 탄창 및 자료정보를 현시한다. 사실 【Threads, call stacks and data】보기에서는 프로그람의 모든 스레드묶음에 대하여 그것들의 현재 상태를 현시하고있다. 이것 역시 메쏘드의 호출순서에 따라 모든 메쏘드들을 현시한다. 이 보기를 리용하여 구체적인 호출과정을 추적할수 있으며 현재의 오유까

지 찾아갈수 있다. 물론 이 창문에서 메쏘드를 호출하는곳까지 돌아올수도 있다. 매 스레드묶음은 자기의 스레드들을 현시할뿐아니라 현재의 메쏘드가 호출한 탄창프레임궤적(stack frame trace)을 포함하고있다. 매 탄창구조에서는 실행중에 얻을수 있는 자료요소들을 현시하고있다.

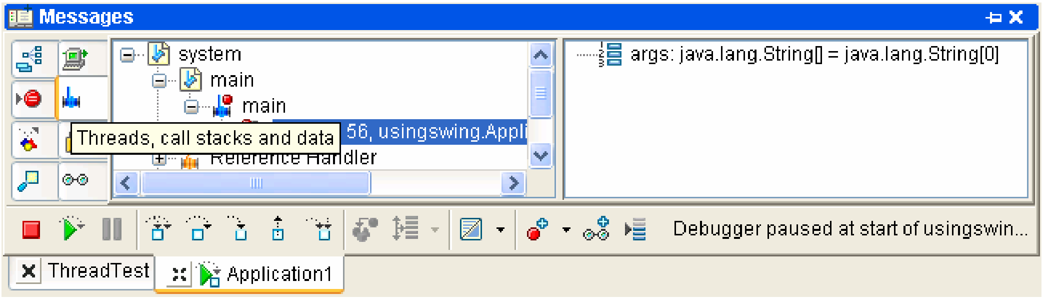


그림 2-29. 스레드, 메쏘드가 호출하는 탄창과 자료정보창문

【Threads, call stacks and data】창문에서 마우스오른쪽찰칵하면 지름차림표가 나타난다(그림 2-30). 여기서 【Change Value】항목을 선택하면 【Change Value】대화칸이 현시된다.(그림 2-31) 이 대화칸에서 String의 값이나 임의의 단순자료류형을 직접 편집할수 있는데 이 류형값에는 수자와 Booleans값들이 포함된다.



단순자료류형으로 선택되여야만 이것을 사용할수 있다.

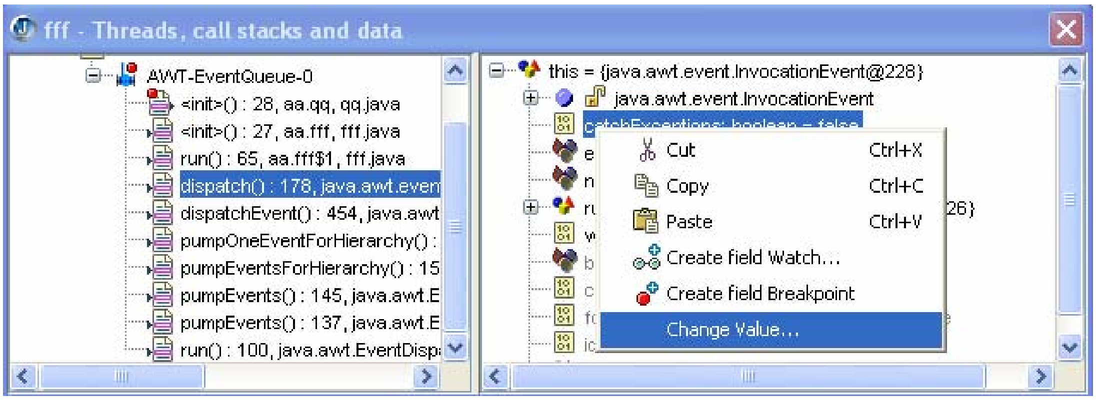


그림 2-30 Threads, call stacks and data창문에서 오른쪽단추를 찰칵하였을 때 화면

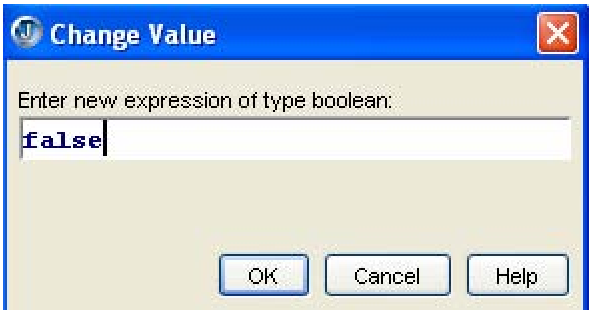


그림 2-31. Change Value대화칸

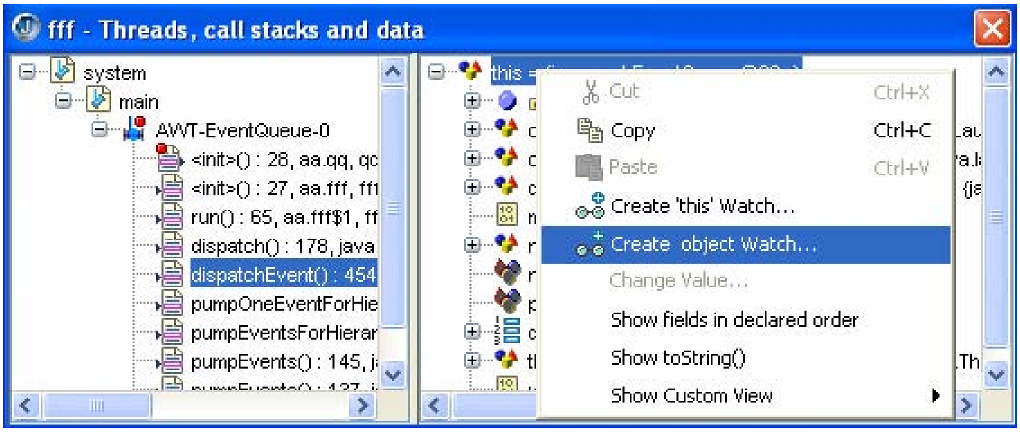


그림 2-32. Create Object Watch항목의 선택



그림 2-33. Add Watch대화칸

【Create Object Watch】차림표항목을 선택하면 【Add Watch】대화칸이 열린다.(그림 2-32,2-33) 이 대화칸에서 선택된 객체에 대하여 감시(watch)를 창조한다. 이 감시는【Data watchs】보기에 추가하여 들어간다. 【Create Field Breakpoint】차림표항목을 선택하면【Add Field Breakpoint】대화칸이 열리며 선택된 마당에서 중단점을 창조한다.

그림 2-34에서 보여주는것은 자료변수와 객체를 관측하는 창문이다.

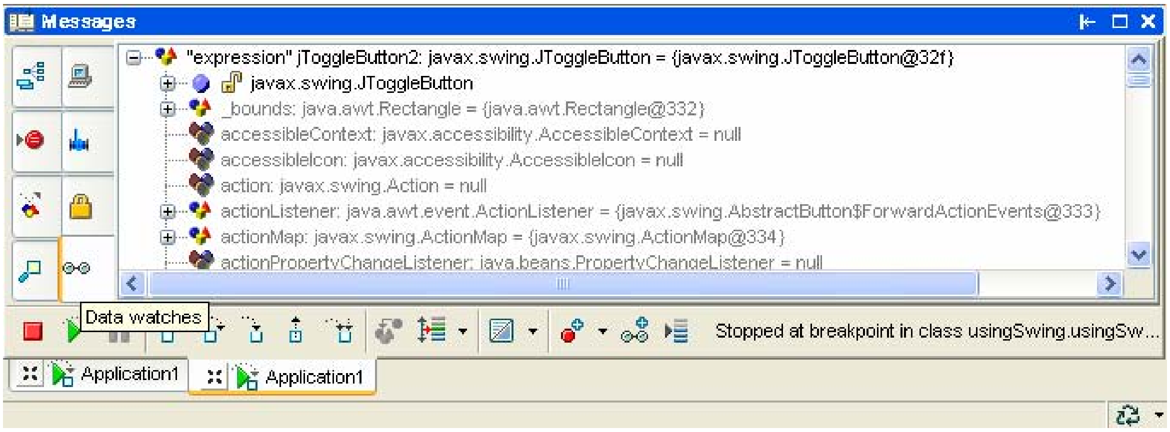


그림 2-34. 자료관측창문

우선 관측할 객체를 추가하여야 한다. 여기서는 【Run】차림표의 【Add Watch...】지령을 찰칵하여 관측할 객체 jToggleButton2을 추가한다. 다음에 이 객체에 대한 중단점을 찾는다. 그림 2-30에서 사용한것은 usingSwingFrame.java프로그람의 148행이다. 【Run】차림표의【Debug Project】를 실행하여 오유제거상태에 들어가면 그림 2-34에서 보여주는 결과를 얻을수 있다.

이 그림에서 jToggleButton2은 javax.swing.jToggleButton의 부품이라는것을 보여주고있다. 【+】단추를 찰칵하면 이 항목이 전개된다. 그리고 jToggleButton2의 일부 변수들에 대하여 그것들이 실행될 때의 값들을 관측할수 있다.

제3장. JBuilder의 다중스레드기술

Java언어의 목적의 하나는 지금 리용하고있는 조작체계들에서 스레드관리를 실현할수 있는 기초를 확립하자는데 있다. Java가상기계의 대부분의 과제들은 스레드관리에 의존하고있으며 클라스서고는 다중스레드관리의 대표적인 실례라고 할수 있다. 사용자가 스레드를 처리하는데서 대부분의 시간은 사용자가 건반을 치거나 마우스를 찰칵하기를 기다릴 때이므로 다중스레드처리는 중요한 문제로 나선다.

제1절. 다중스레드개념

아래에서 다중스레드에 대한 개념을 간단히 소개한다.

3.1.1. 다중스레드란

다중스레드 프로그람작성에서는 다음의 단계를 거친다.

먼저 프로그람과제를 몇개의 병렬적인 부분과제들로 분할한다. 이러한 문제는 망프로그람작성에서 제기된다. 실례로 망상에서 자료전달속도가 뜨고 사용자입력속도가 뜰 때에 이것들을 두개의 독립적인 스레드로 하여 두개의 기능을 완성하면 정상적인 현시와 다른 기능에 영향을 주지 않을수 있다.

다중스레드와 달리 단일스레드프로그람구조는 보통의 Windows에서 많이 리용하고있는데 그 동작원리는 다음과 같다.

주프로그람은 하나의 통보순환을 가지고 부단히 통보기다림렬로부터 통보를 읽어들여 다음 단계에서 해야 할 사업을 결정한다. 실례로 하나의 망자료를 읽거나 파일을 읽는데서 오직 이 자료들이나 파일들을 다 읽어 들인 다음에야 다음 통보를 접수할수 있다. 그러므로 망자료를 읽고 사용자가 입력하는 시간동안 기다림상태에 놓이게 된다.

이런 경우에는 다중스레드를 리용하여 과제를 여러개로 갈라 놓고 문제를 해결해야 한다. Java에서는 이것을 해결할수 있도록 스레드지원기능을 내장하고있다. 다중스레드는 사용자가 동시에 여러가지 작업을 할수 있게 한다. 만일 한사람이 같은 시간에 팔이나 다리를 동시에 움직이게 된다면 이 사람은 틀림없이 제한을 적게 받는다. 다중스레드란 바로 이와 류사한 문제들을 콤퓨터에서 해결할수 있도록 하는것이라고 말할수 있다.

다중스레드의 개념을 업무과정을 놓고 리해하도록 하자. 대부분의 기업소들은 적어도 3개의 독립적인 부분을 가지고있다. 즉 관리부, 회계부, 생산 및 영업부이다. 기업소가 잘 운영되자면 3개 부분이 동시에 움직여야 한다. 만일 회계부의 사업이 멈춰서면 기업소는 문을 닫을수 있으며 관리부문이 움직일수 없으면 기업소는 해산될수 있다. 그리고 생산부문

이 활동할수 없으면 기업소는 자금을 확보할수 없다.

쏘프트웨어 역시 기업소와 꼭같은 조건에서 동작한다. 기업소에서는 과제를 서로 다른 사람들에게 분담하여 완성하지만 쏘프트웨어에서는 일반적으로 사용자가 처리기를 가지기만 하면 처리기는 모든 과제를 다 접수하여 처리한다. 이 처리기를 관리하기 위하여 《다중과제》의 개념이 생겨났다. 처리기는 실제상 같은 시간에 하나의 일감만을 처리하지만 시분할방식으로 과제를 분할하여 과제들사이의 고속절환으로 다중과제를 처리함으로써 사람들이 처리기가 동시에 여러가지 일감을 처리하는것처럼 느끼게 한다.

3.1.2. 다중스레드를 사용하는 리유

다른 프로그람작성언어에 비한 Java언어의 가장 큰 우점은 그것이 다중스레드를 지원하는 기능을 내장하고있는것이다. 스레드를 사용하여 사용자는 과제실행과 결과보기사이에 기다리는것을 피할수 있으며 인쇄작업을 후에 할수도 있고 먼저 문서를 입력하거나 어떤 다른 과제들을 집행할수도 있다.

Java에서 스레드를 리용하여 보통 Applet가 뒤에서 어떤 일감을 수행하게 하고 열람기는 자기의 작업을 계속하게 할수 있다.

많은 쏘프트웨어문제들중에서 다중스레드를 리용하여 해결할수 있는것들이 많다. 실례로 도형방식으로 자료를 현시하는 대화식프로그람은 보통 실시간적으로 파라메터에 대한 현시가 변경된다. 이것은 스레드를 리용하여 쉽게 실현할수 있다.

스레드의 사용은 대화식프로그람에서 가장 좋은 동적효과를 얻을수 있게 한다. 단일스레드체계에서는 일반적으로 새치기나 송신문에 대하여 한개 응용프로그람에 현시부분과 사용자입력부분을 함께 혼합하여야 한다. 이 경우 현시부분에서 사용자의 순간적인 입력에 즉시 응답하도록 프로그람을 작성하여야 한다. 이것은 스레드의 처리에서 복잡성을 가져온다.

그러나 다중스레드체계에서는 이러한 문제들을 쉽게 해결할수 있다. 즉 한개의 스레드는 현재의 자료를 수정현시하고 다른 스레드는 사용자의 입력에 응답하도록 하면 된다. 그렇게 되면 사용자입력이 비교적 복잡하다고 하여도(실례로 표에 자료를 써넣는 경우) 현시프로그람은 독립적으로 실행할수 있으며 새로운 자료를 계속 접수할수 있게 된다.

3.1.3. 다중스레드의 실례

은행에서의 저금봉사를 실례로 다중스레드의 개념을 구체적으로 설명한다. 순차실행프로그람에서 이 처리과정을 보면 그림 3-1과 같이 이전의 부족수를 가지고 저금수를 추가한다. 그리고 마지막에 돈자리를 기록보관한다.

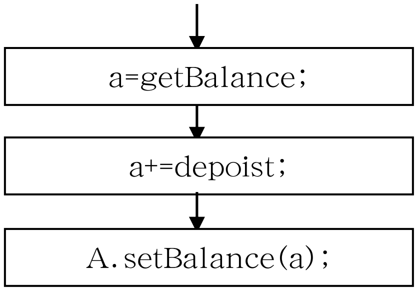


그림 3-1. 단일스레드처리과정

실제의 은행업무와 콤퓨터프로그람은 류사한 집행순서를 가지고있다. 단일스레드프로그람작성모형은 많은 프로그람작성자들이 일부 다른 프로그람작성언어에서 익숙한 형식이다.

실제의 은행업무에서는 여러 조작을 동시에 진행하여야 하는 경우가 있다. 즉 은행직원들이 은행돈자리를 독자적으로 수정할수 있다.(그림 3-2)

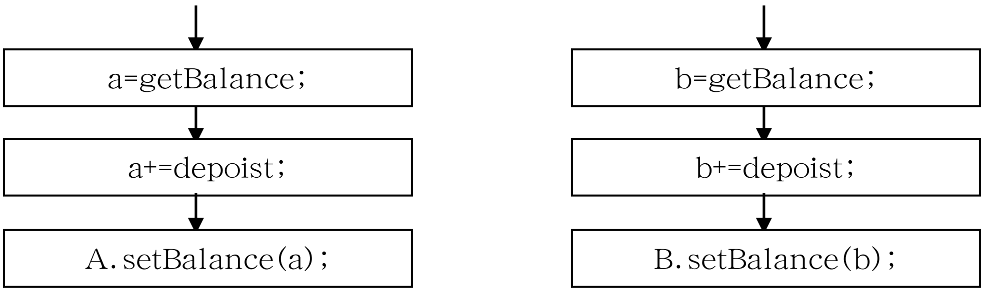


그림 3-2. 다중스레드처리과정

콤퓨터에서도 은행업무에서와 같은 다중스레드가 있다. 한개의 스레드는 다른 스레드와 독립적으로 한개의 과제를 처리한다. 두명의 출납이 같은 은행자료를 사용할수 있는것과 같이 스레드들은 객체에 대한 접근을 동시에 할수 있다.

제2절. 스레드의 창조

이 절에서는 스레드의 창조과정에 대하여 설명한다.

3.2.1. Thread클라스를 사용한 창조

스레드는 표준적인 Java서고의 클라스로부터 정의한다. 조종스레드를 창조하려면 우선 스레드객체를 창조하여야 한다. 즉

Thread newthread=new Thread();

스레드객체가 창조되면 그것에 대하여 설계를 한 다음 실행한다. 한개 스레드객체에는 초기화우선권, 이름 등이 포괄되여있다. 스레드가 실행될 때 먼저 그의 start메쏘드가 호출된다. start메쏘드는 스레드객체의 자료에 따라 새로운 조종스레드를 생성한 다음에 귀환한다. 계속하여 start는 새로운 스레드의 run메쏘드를 호출하여 이 스레드를 능동으로 한다.

스레드의 run메쏘드가 귀환될 때 스레드역시 저절로 탈퇴된다. 스레드의 stop메쏘드를 호출하여 스레드를 중지할수 있다. 또한 suspend메쏘드를 리용하여 스레드의 집행을 보류할수도 있다.

Thread.run을 그대로 사용할수 없다. 사용자는 Thread를 계승하여 새로운 run메쏘드를 제공하거나 Runnable객체를 창조하고 이것을 스레드의 구성자에게 전달하여야 한다.

여기서는 우선 Thread클라스를 리용하여 새로운 스레드를 창조하는 방법을 보고 다음 절에서 Runnable대면을 리용한 실현수법을 보기로 한다.

Java에서 새로운 스레드를 창조하는것은 간단하다. 단지 java.lang.Thread클라스, 그것을 재정의하는 run메쏘드를 확장하면 된다.

실례 3-1

Thread클라스를 사용하여 새로운 스레드를 창조한다.

1. Project Wizard를 리용한 프로젝트의 창조

Project Wizard를 리용하여 프로젝트를 창조하는 구체적인 단계는 다음과 같다.

단계

1 【File】→【New Project】지령을 선택한다.

2 【Name:】본문칸에서 《untitled》를 《ThreadTest》로 수정한다.

3 【Directory:】복합칸에서 요구하는 작업등록부를 선택한다. 다음에 【Next】를 찰칵하여 다음 페지로 들어간다.

4 이 페지에서 각종 환경설정을 수정할수 있는데 이 실례에서는 기정으로 설정하고【Next】를 찰칵하여 다음 페지로 들어간다.

5 【Encoding:】복합칸에서 《Big 5》를 선택한다. 【Description:】마당에 《This is a thread test program》을 추가한다. 【Title:】마당에 《Thread test》를 입력한다.

6 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

Application Wizard를 리용하는 단계는 다음과 같다.

단계

1 【File】→【New】지령을 선택하면 New대화칸이 나타난다

2 Application아이콘을 찰칵한다. 이때 Application Wizard대화칸이 나타난다.

3 【Class name:】본문칸에 《Threadtest》를 입력한다.

4 【Next】단추를 찰칵한다.

5 【Title:】마당에 《Thread test》를 입력한다.

6【Finish】단추를 찰칵한다.

3. 프레임에 조종부품추가

프레임에 조종부품을 추가하는 단계는 다음과 같다.

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택하고 Design태브를 찰칵한다.

2 부품선택판에서 AWT항목을 찰칵하고 java.awt.TextArea부품을 선택한다.

3 Design에 TextArea조종부품을 그린다.

4 Source태브를 찰칵하여 private void jbInit()메쏘드의 《textAreal.setText(“textAreal”);》을 삭제하고 다음의 코드내용을 추가한다.

```
class ThreadCount extends Thread
{
    private int start_num;
    private int end_num;
    public ThreadCount(int from,int to)
```

```
    {
        this.start_num=from;
        this.end_num=to;
    }
    public void run()
    {
        textArea1.append((this.getName()+"started executing...\n"));
        for(int i=start_num;i<=end_num;i++)
        {
            textArea1.append(i+" ");
        }
        textArea1.append((this.getName()+"finisfed executing.\n"));
     }
}
ThreadCount thread1=new ThreadCount(1,10);
ThreadCount thread2=new ThreadCount(20,30);
thread1.start();
thread2.start();
}
```

5 【File】→【Save All】지령을 선택하여 완성한 작업을 보관한다.

6 【Run】→【Run Project】지령을 선택하여 콤파일과 실행을 진행한다.

실행결과는 그림 3-3과 같다.

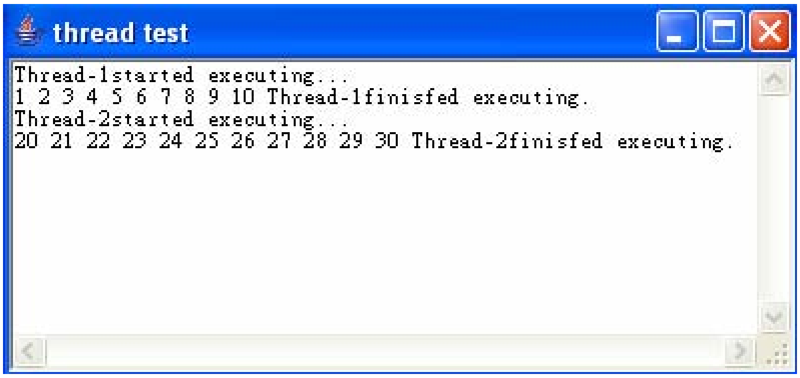


그림 3-3. 실례 3-1의 실행결과

출력되는 스레드이름은 thread1과 thread2가 아니다. 왜냐하면 스레드의 이름을 특별히 지적하지 않는한 Java는 자동적으로 스레드의 이름을 《Thread-n》의 형식으로 만들어 준다. 여기서 n은 0을 포함한 정의 옹근수값을 가진다. 그러므로 구성자에서 스레드에 대하여 한개의 이름을 지정하거나 또는 setName(String)메쏘드를 사용하여 스레드이름짓기를 할수 있다.

이 실례에서 Thread-0은 제일 먼저 집행되여 제일 먼저 결속된다. 그러나 어떤 경우에는 제일 먼저 집행되여 제일 마지막에 결속될수도 있고 집행도중에 Thread-1에 의해 중단될수도 있다. 이것은 Java에서 스레드가 확정적인 집행순서를 가지고있지 않기때문이다. 매번 스레드검사를 진행하여 얻은 결과는 완전히 같을수 없다. 스레드집행의 일정짜기는 프로그람작성자가 정하는것이 아니라 Java스레드일정짜기기구(Java thread scheduler)에 의해 작성된다.

3.2.2. Runnable대면

스레드를 창조하는 다른 하나의 간단한 방법은 java.lang.Runnable대면을 사용하는것이다. Runnable은 하나의 코드집행단위를 추상화한것이다.

Runnable대면을 실현하는 메쏘드를 리용하여 매개 객체의 스레드를 창조할수 있다. 례를 들어 우의 실례에서 클라스 ThreadCount는 다음과 같이 정의할수 있다.

public class ThreadCount implements Runnable{

Runnable대면을 실현하기 위하여 어떤 하나의 클라스는 반드시 run()메쏘드를 실현하여야 한다. 이 메쏘드선언은 다음과 같다.

public void run()

run()에서 코드를 정의하면 새로운 스레드를 만들수 있다. 문제는 아래의 내용을 리해하는것이 중요하다. 즉 run()메쏘드는 주스레드와 같이 다른 메쏘드를 능히 호출할수 있으며 다른 클라스를 인용하여 변수를 선언할수 있다. 차이점은 run()이 프로그람에서 스레드집행입구를 갖추고있다는것이다. run()이 귀환될 때 이 스레드가 결속된다.

Runnable대면을 실현하는 클라스를 창조한 다음에는 클라스안에서 Thread클라스의 객체를 구체례로 하여야 한다. Thread클라스는 여러가지 구성자를 정의하고있다. 구성자에는 Thread(), Thread(Runnable target), Thread(Runnable target, String name), Thread(String name), Thread(ThreadGroup group, Runnable target), Thread(ThreadGroup group, Runnable target, String name), Thread(ThreadGroup group, Stringname) 등을 포괄하고있다.

새로운 스레드를 작성한 다음에는 그것의 start()메쏘드를 호출하여 실행하여야 한다. 이

메쏘드는 Thread클라스에서 정의한다. 사실상 start()메쏘드에 대한 호출은 곧 run()에 대한 호출이다. start()메쏘드는 다음과 같이 선언한다.

void start()

실례 3-2

새로운 스레드를 창조하고 그것을 기동하여 실행해보자. JBuilder에서의 설계단계는 다음과 같다.

1. Project Wizard를 사용한 프로젝트의 창조

Project Wizard를 사용하여 프로젝트를 창조하는 단계는 다음과 같다.

단계

1 【File】→【New Project】지령을 선택한다.

2 【Name:】본문칸에서 《untitled》을 《ThreadTest》로 수정한다.

3 【Directory:】복합칸에서 작업에 필요한 등록부를 선택한다. 다음 【Next】를 찰칵하여 다음 페지로 넘어간다.

4 이 페지에서 각종 환경설정을 수정한다. 여기서는 기정으로 설정하고 【Next】를 찰칵하여 다음 페지로 넘어간다.

5 【Encoding:】복합칸에서 《Big 5》를 선택한다. 【Description:】마당에 《This is a thread demo program》을 기입한다.

【Title:】마당에 《create thread use runnable interface》를 입력한다.

6 【Finish】단추를 찰칵한다.

2. Application Wizard 사용

Application Wizard를 사용하는 단계는 다음과 같다.

단계

1 【File】→【New】지령을 선택하면 New대화칸이 나타난다.

2 Application아이콘을 찰칵한다.

이때 Application Wizard대화칸이 나타난다.

3 【Class name:】본문칸에서 《ThreadTest》를 입력한다.

4 【Next】단추를 찰칵한다.

5 【Title:】본문칸에 《create Thread using runnable interface》를 입력한다.

6 【Finish】단추를 찰칵한다.

3. **프레임에 조종부품추가**

프레임에 조종부품을 추가하는 단계는 다음과 같다.

단계

1 프로젝트판의 Navigation에서 Frame1.java를 설정하고 다음에 Design태브를 찰칵한다.

2 부품선택판에서 AWT를 찰칵하고 java.awt.TextArea조종부품을 선택한다.

3 Design방식에서 TextArea조종부품을 그려넣는다.

4 Source태브를 찰칵하고 private void jbInit()메쏘드의 《textArea1.setText("text Areal");》을 삭제하고 아래의 코드내용을 입력한다.

```
class ThreadRunnableDemo implements Runnable
{
    String name;
    ThreadRunnableDemo(String s)
    {
        name=s;
    }
    public void run()
    {
        textArea1.append(name+"started executing...\n");
        for(int i=0;i<10;i++)
        {
        textArea1.append(i+" ");
        }
        textArea1.append(name+" finished executing.\n");
    }
}
ThreadRunnableDemo Demo1=new ThreadRunnableDemo("Thread1");
ThreadRunnableDemo Demo2=new ThreadRunnableDemo("Thread2");
Thread t1=new Thread(Demo1);
Thread t2=new Thread(Demo2);
t1.start();
t2.start();
}
```

프로그람실행결과는 그림 3-4와 같다.

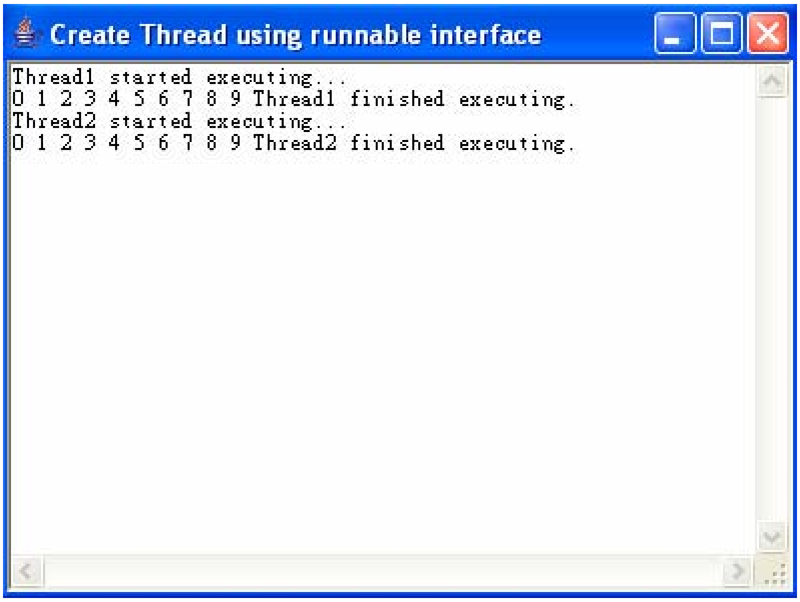


그림 3-4. Runnable대면을 사용하여 스레드를 창조

실례 3-1과 다른점은 이것의 스레드이름들이 다른것이다. 그것은 실례 3-2에서는 구성자메쏘드를 리용하여 스레드이름을 지정하고있기때문이다.

제3절. 스레드의 관리

아래에서 스레드의 관리에 대하여 소개한다.

3.3.1. 스레드의 상태와 그의 전환

지금까지는 자동적으로 완료되는 스레드를 작성하였다. 이 스레드들은 한가지 과제를 수행한 다음에는 완료된다. Java프로그람에서 이 스레드들을 어떻게 중지하는가는 아주 중요한 문제이다.

아래의 단일스레드프로그람사용에 대하여 생각해보자. 이러한 프로그람에서는 모든 스레드들이 완료될 때에야 종결된다. 그러므로 한개의 스레드가 실행을 끝마치기전에는 프로그람은 영원히 완료될수 없다.

매개 스레드는 4가지의 상태를 가진다. 즉 new(새로 창조), running(실행), waiting(대기), done(완료)을 가지며 스레드는 오직 이것중의 한가지 상태에만 놓일수 있다.

스레드가 처음으로 창조될 때에는 new상태에 놓인다.

new상태에서는 스레드가 실행되지 못한다. 즉 이 상태에서 스레드들은 기동을 위해 기다린다. 스레드는 start메쏘드를 사용하여 기동하거나 stop메쏘드를 리용하여 정지(이때에는 스레드가 done상태로 된다.)된다. 실행이 끝난 상태에 놓인 스레드는 결속된다. 이것은 스레드의 마지막상태이며 일단 스레드가 이러한 상태에 놓이면 실행상태로 다시 회복될수 없다. 또한 모든 스레드가 다 완료상태에 놓이면 프로그람은 실행을 정지한다. 현재 집행하고있는 모든 스레드들은 거의 실행상태에 있으므로 처리기는 어떤 방식에 따라 스레드사이에서 분산처리를 한다. 실행상태에 놓인 스레드는 실행될수 있지만 임의의 시각에 체계처리기는 오직 하나의 프로그람만을 실행할수 있다. 대기상태는 스레드의 집행이 중단된 경우에 놓이는 상태이다. 하나의 스레드는 여러가지 방식으로 중단될수 있다. 이때 이것은 보류(suspend)되여 체계자원의 분배를 기다리게 되거나 잠자기(sleeping)에 들어가게 된다. 이렇게 보류되였을 때 스레드는 대기상태로부터 실행상태로 전환되거나 stop메쏘드를 리용하여 완료상태에 놓이게 된다. 표 3-1은 스레드를 조종하는 메쏘드들을 보여주고있다.

표 3-1. 스레드조종메쏘드

메쏘드	설명	현재상태	다음상태
start()	스레드집행시작	새로 창조	실행
stop()	스레드집행정지	새로 창조, 실행	완료
sleep(long)	잠자기시간을 미리 초단위로 지정	실행	대기
sleep(long,int)	잠자기시간지정	실행	대기
suspend()	스레드보류	실행	대기
resume()	계속집행	대기	실행
yield()	석방조종	실행	실행

3.3.2. isAlive()와 join()

표 3-1의 메쏘드들은 모든 시각에 다 유효한것이 아니며 대부분의 메쏘드들은 스레드가 실행상태에 놓일 때 리용할수 있다. 만일 스레드의 상태에 맞지 않는 어떤 메쏘드를 사용한다면 실례로 쓸모없이 된 스레드를 보류한다면 IllegalThreadStateException례외가 발생할수 있다. 일반적으로 프로그람작성자는 개발과정에 현재 스레드가 어느 상태에 있는가를 똑똑히 알아야 한다. 스레드의 상태를 결정하는 두가지 메쏘드가 있는데 이것으로 스레

드가 완료상태인가를 판정할수 있다. 하나는 isAlive()메쏘드이다. 이 메쏘드는 Thread에 의해 정의하며 그것의 일반형식은 다음과 같다.

final boolean isAlive();

만일 호출한 스레드가 여전히 실행상태에 있으면 isAlive()메쏘드는 true를 귀환하고 그렇지 않으면 false를 귀환한다. 그러나 isAlive()는 매우 적게 쓰인다. 일반적으로는 join()메쏘드를 사용한다.

final void join() throws InterruptedException;

이 메쏘드는 호출한 스레드가 완료되기를 기다린다. 아래의 실례는 실례 3-1과 3-2를 참고하여 작성한것이다.

```
class DemoThread implements Runnable
{
    String name;// thread name
    Thread t;

    DemoThread(String Name_of_Thread){
        name=Name_of_Thread;
        t=new Thread(this,name);
        textArea1.append("New thread: "+t+"\n");
        t.start(); // start thread
   }
   public void run()
   {
       try{
           for(int i=4;i>0;i--)
           {
               textArea1.append(name+": "+i+"\n");
               Thread.sleep(1000); //sleep 1 second
           }

       }
       catch(InterruptedException e){
           textArea1.append(name+"interrupted.\n");
       }
   }
}
```

```
DemoThread th1=new DemoThread("One");
DemoThread th2=new DemoThread("Two");

textArea1.append("Thread One is alive: "+th1.t.isAlive()+"\n");
textArea1.append("Thread Two is alive: "+th2.t.isAlive()+"\n");
// Waiting for threads to finish
try{
    textArea1.append("Waiting for threads to finish.\n");
    th1.t.join();
    th2.t.join();
}
catch(InterruptedException e){
    textArea1.append("Main thread Interrupted\n");
}
textArea1.append("Thread One is alive: "+th1.t.isAlive()+"\n");
textArea1.append("Thread Two is alive: "+th2.t.isAlive()+"\n");
textArea1.append("Main thread exited.");
}
```

프로그람의 실행결과는 그림 3-5에서 보여준다.

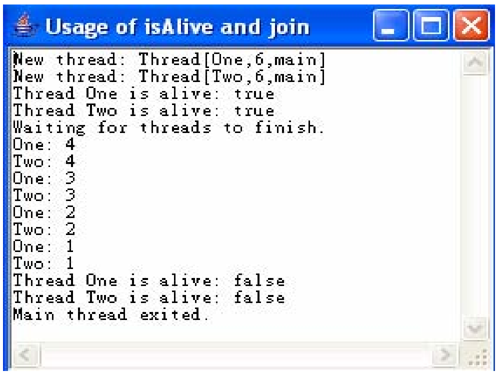


그림 3-5. isAlive()와 join()의 사용

3.3.3. 스레드 일정짜기

스레드를 실행시키는 순서와 그것들의 처리시간은 개발자가 관심하는 기본문제이다. 매개 스레드는 처리기시간내에 처리된다. 여러개의 스레드가 있을 경우에는 일정짜기를 하여

야 한다. 스레드는 일정짜기의 두가지 개념 즉 선매권(preemption), 시간토막(time slicing)과 밀접한 관계를 가진다. 선매권이란 통속적으로 말하여 먼저 살 권한을 가진다는것을 의미한다. 스레드의 일정짜기에서는 처리의 우선권을 가진다는것을 의미한다. 보통 일정짜기프로그람들은 두가지 일정짜기방안 즉 비선매권일정짜기와 선매권시간토막 일정짜기에 따라 작업한다.

비선매권일정짜기를 사용하면 프로그람은 현재의 스레드를 끝까지 실행한 다음 이 스레드가 일정짜기프로그람에 알려져서 다른 스레드를 시작할수 있을 때에야만 집행권한을 포기한다. 반면에 선매권시간토막 일정짜기방법에서는 프로그람이 현재의 스레드를 실행하여 이 스레드가 어떤 시간토막(time slicing)동안에 실행될 때까지 다른 스레드들은 기다리게 된다. 사용시간이 지나면 이 스레드는 보류상태에 들어가며 다른 스레드가 회복되여 다음 시간토막을 사용하게 된다. 일반적으로 비선매권일정짜기는 실시간적인 요구성이 높은 프로그람에서 리용한다. 그것은 시간토막의 한정값에 따르는 스레드의 새치기가 실시간처리에서 엄중한 손실을 가져올수 있기때문이다.

대다수의 일정짜기프로그람들은 선매권시간토막방안을 리용한다. 그것은 일부 실시간적인 요구가 높은 경우를 제외하고는 이 일정짜기방식이 보다 쉽게 다중스레드프로그람을 작성할수 있기때문이다. 이 일정짜기프로그람은 모든 스레드에 공평하게 동일한 실행시간을 배정한다. 현재의 많은 일정짜기프로그람들에서는 스레드우선권개념을 부가하여 우선권에 따라 스레드순서를 정한다. 보통 우선권은 스레드의 중요성을 나타낸다. 우선권이 높은 스레드는 상대적으로 많은 실행기회를 얻을수 있고 상대적으로 많이 새치기할수 있다는것을 의미한다.

Java에서 모든 스레드는 한개의 우선권을 가질수 있으며 스레드창조시 이것은 그것을 창조하는 스레드로부터 우선권을 계승한다. 클라스변수 Thread.NORM_Priority는 스레드의 기정값이다. Thread클라스는 setPriority와 getPriority메쏘드로 우선권을 설정하고 우선권을 얻는다. setPriority메쏘드를 사용하면 Java가상기계에서 스레드의 우선권을 변경시킬수 있다. 이 방법은 다음과 같다.

```
final void setPriority(int level);
```

setPriority는 옹근수파라메터 level을 가진다. 이 값은 두개의 클라스변수 Thread.MIN_PRIORITY와 Thread.MAX_PRIORITY 사이의 값을 가진다. 보통 이 값들은 각각 1과 10이다. 스레드에 기정인 우선권을 주려면 NORM_PRIORITY를 지정하여 주는데 이 값은 보통 5이다. 이 우선권들은 Thread에서 final형변수로 정의된다. 또한 Thread의 getPriority()메쏘드를 호출하여 현재의 우선권값을 얻을수도 있다. 이 방법은 다음과 같다.

```
final int getPriority();
```

리론적으로 보면 우선권이 높은 스레드가 우선권이 낮은 스레드에 비하여 더 많은 CPU 시간을 가지게 된다. 실제상 스레드가 가지게 되는 CPU시간은 보통 우선권과 관련된 많은 인자들에 의하여 결정된다. 우선권이 높은 스레드는 응당 우선권이 낮은 스레드에 비하여 우선적이다. 실례를 들어본다면 우선권이 낮은 스레드가 현재 실행중에 있는 상태에서 우선권이 높은 스레드가 회복(례하면 잠자기중이거나 I/O기다림상태에서)될 때 우선권이 낮은 스레드에서 사용하고있는 CPU를 가로채서 리용하게 된다. 그밖에 리론상 우선권이 같은 스레드들은 동등한 권리를 가지고 CPU를 리용한다. 그러나 Java는 여러 환경에서 동작할수 있으므로 매 환경에서 과제처리는 약간한 차이를 가지게 된다.

3.3.4. 스레드그룹

적은 개수의 스레드들을 관리하는데서는 큰 문제가 생기지 않는다. 그러나 만일 처리하여야 할 스레드가 수백개인 경우에는 관리에서 많은 문제들이 제기된다. 례하면 Internet 봉사기측프로그람을 작성할 때에 매개 스레드들을 반복처리하여야 할 문제들이 생긴다. 이때 어떤 봉사기들은 수천번의 대화과정을 거쳐야 한다. 이 경우 체계는 모든 대화과정을 다 처리하여 결속하여야 한다. 이때 두가지 방안 즉 매 스레드에 대한 순환처리와 그것의 완료, 혹은 스레드그룹(thread groups)처리가 있다.

스레드그룹은 계층구조를 가진 스레드들의 집합이다. 매개 그룹은 개수에 제한없이 많은 스레드들을 포함할수 있으며 매개 스레드에 대한 이름짓기와 일부 조작(실례로 그룹에 대한 완전한 보류와 정지 등)들을 집행할수 있다.

다음과 같이 스레드그룹을 창조한다.

ThreadGroup parent=new ThreadGroup(“parent”);

ThreadGroup child=new ThreadGroup(parent, “child”);

이렇게 스레드그룹을 창조하는데는 두가지 메쏘드가 있다. 첫째 메쏘드는 스레드그룹을 직접 창조하는것이고 둘째 메쏘드는 상위그룹과 이름을 리용하여 창조하는것이다. 스레드의 상위그룹은 그것이 지령할수 있는 매개 스레드에 영향을 준다.

일단 스레드그룹의 객체가 창조되면 새로운 스레드를 그룹에 추가할수 있다. 스레드그룹에서 사용하고 처리할수 있는 Thread구성자를 리용하여 새로운 스레드를 추가할수 있다. 스레드그룹은 창조후에 수정할수 없다. 스레드그룹에 새로운 스레드를 추가하는 방법은 다음과 같다.

Thread t1=new Thread(parent);

Thread t2=new Thread(child, “t2”);

스레드그룹에 제일 유용한 메쏘드는 stop, suspend, resume이다. 스레드그룹에서의 매개 스레드들은 자기의 호출가능한 메쏘드들을 가진다. 스레드그룹의 하위스레드 역시 이에 따르는 영향을 받는다. 그러므로 이 메쏘드들을 사용하면 쉽게 많은 스레드들을 조작할 수 있다.

스레드그룹에는 또한 다른 기능들도 있다. 스레드그룹에서는 Java가동기반에서 스레드들사이의 안전성문제를 잘 처리하여야 한다. 일반적으로 Java가상기계에서는 사용자스레드가 체계스레드에 해를 주지 말아야 한다. ThreadGroup의 하위스레드는 상위그룹의 스레드에 명령하고 지시할수 없으므로 사용자가 이러한 류형의 처리를 함부로 할수 없도록 Java가상기계는 이러한 스레드그룹처리를 리용하여 체계스레드와 사용자스레드를 서로 간섭할수 없도록 분리시켜야 한다.

제4절. 스레드의 동기화

이 절에서는 스레드의 동기화에 대하여 소개한다.

3.4.1. 스레드동기화의 개념s

동기화란 동시에 실행되고있는 여러개의 스레드들에 대하여 실행시간을 맞춘다는것을 의미한다. 시간맞추기란 《동기를 맞춘다》는 의미이다. 일반적으로 같은 자원을 여러개의 스레드가 동시에 접근하는 프로그람들에서는 동기화할 필요가 제기된다. 여기서 자원(resource)이란 파일이나 기억기 등 콤퓨터의 자원을 말한다. 두개의 스레드가 꼭같은 자료를 공유하여 수정하는 경우 객체상태를 손상시킬수 있다. 이러한 문제를 예방하기 위한 수단이 동기화이다. 동기화는 아래의 그림과 같은 순차방식으로 실현한다.(그림 3-6)

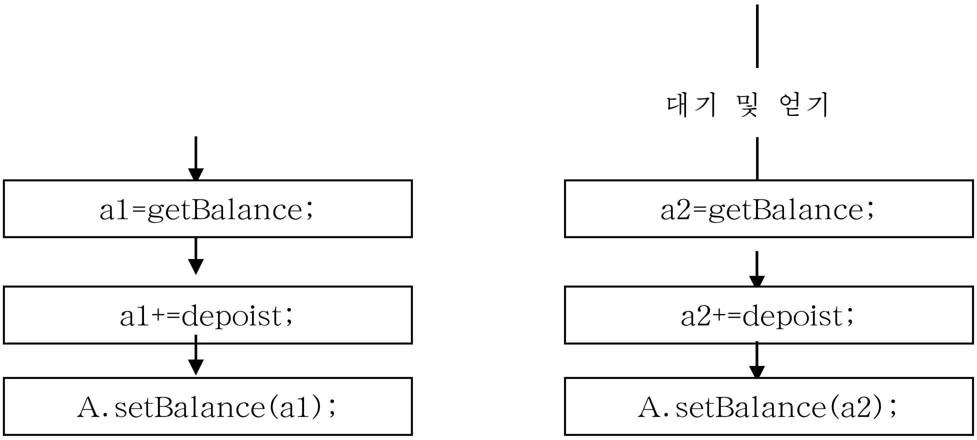


그림 3-6. 두 스레드의 같은 대상에 대한 처리과정

동기화가 필요한 실례로서 은행에서 계산자리(구좌라고 함)에서의 입금처리과정을 보기로 보자. 앞에서 본 은행업무실례에서 출납을 통하여 어떤 계산자리에 돈을 넣는 사람이

있다고 하자. 거의 동시에 두번째 손님이 역시 다른 출납에게 동일한 계산자리를 통하여 돈을 넣으려고 한다. 이 경우 두 출납은 자료기지에서 같은 계산자리정보를 얻는다. 다음에 그들은 각자 출납대로 돌아와 저금을 넣고 다시 자료기지에 되돌려 보낸다. 그리고 독립적인 조작의 결과를 각각 기록한다. 만일 이렇게 하면 두번째 손님의 저금기록만이 계산자리의 잔고에 반영되고 첫번째 손님의 저금은 잃게 된다.

은행업무에서 은행자료기지에 두 직원이 동시에 같은 계산자리에로 접근할 때 생기는 이러한 문제는 다음과 같이 해결할수 있다. 두번째 손님의 요청에 다음과 같이 통보한다. 《이 자료는 현재 처리중입니다. 좀 기다려주십시오.》 콤퓨터에서도 역시 이렇게 처리한다. 만일 스레드1이 실행상태에 있다면 스레드2는 스레드1이 끝나기를 기다렸다가 끝난 다음에 실행한다. 이것을 동기라고 한다.

두개이상의 스레드가 자원을 공유하려고 하는 경우 이 스레드들은 어떤 방법을 리용하든지 자원을 어느 시각에 어느 스레드가 차지하고있는가를 확정하여야 한다. 이 목적을 실현하는 과정을 동기화(synchronization)라고 한다.

아래에서는 동기메쏘드 또는 동기명령문를 리용하여 동기화를 실현하는 방법에 대하여 본다. 이때 synchronized라는 열쇠어를 사용한다.

3.4.2. 동기메쏘드

Java에서 동기화는 간단히 실현할수 있다. 메쏘드에 대한 동기화를 실현하려면 메쏘드 이름의 앞에 synchronized라는 열쇠어를 붙여준다. 만일 synchronized를 리용하여 어떤 공유자료객체에 권한설정을 하면 그 객체에 설정을 먼저 요청한 스레드가 없을 경우에는 권한설정을 해준다. 그러나 이미 권한설정이 되여있는 상태이라면 그 요청은 객체의 요청대기실에서 기다려야 한다. 스레드들의 요청이 계속 있게 되면 그 요청들은 계속 줄서 기다리게 된다. 또한 앞서 요청된 권한설정이 풀리게 되면 객체에 권한요청을 한 남아있는 요청들중에서 어느 하나가 선택되여 그 요청에 해당한 스레드에게 권한설정이 된다. 실례로 앞에서의 은행문제의 다중스레드 Account클라스를 고찰하자. 대응하는 코드는 아래와 같다.

```
class Account{
private double balance;
public Account(double initialDeposit){
balance=initialDeposit;
}
public synchronized double getBalance(){ //getBalance()는 동기메쏘드
return balance;
}
public synchronized void deposit(double amount){//deposit()를 동기메
쏘드로 선언
```

```
    balance+=amount;
    }
    }
```

이제 동기화를 진행할 때 synchronized로 선언되여야 하는 메쏘드는 어떠하여야 하는가를 론의하자. 우선 구성자는 동기를 요구하지 않는다. 그것은 구성자가 객체창조시에만 집행되기때문이다.

다음으로 마당에 대하여 보자. 우의 코드에서 balance마당은 동기메쏘드의 귀환값으로 된다. 만일 마당값을 변경시킬수 있다면 스레드가 그 값을 수정하고 있을 때에는 다른 스레드들은 그것을 읽을수 없다. 그것은 마당의 접근은 동기를 요구하기때문이다. 그러므로 하나의 스레드가 읽기중인 경우에 다른 스레드가 수정중이면 읽어들인 값은 무효로 된다. synchronized선언이 있으므로 두개 이상의 스레드들사이에 실행상 간섭이 있을수 없다. 그러므로 읽기조작이 먼저 시작된다면 쓰기조작시작전에 읽기조작이 이미 완료되여야 한다.

아래에서 실례를 통하여 동기메쏘드의 작용을 설명한다.

실례 3-3

동기메쏘드를 사용하여 동기를 실행한다. 단계는 실례 3-1을 참고하십시오.

상응한 위치에 아래의 코드를 추가한다.

```
class Synchronize_Print
{
   synchronized void print(String msg)
   {
       textArea1.append(" ["+msg);
       try{
           Thread.sleep(1000); //sleep 1 second
       }
       catch(InterruptedException e){
           textArea1.append("Interrupted");
       }
       textArea1.append("] \n");
   }
}
class Print_Demo implements Runnable
{
```

```
    String msg;
    String ThreadName;
    Synchronize_Print target;
    Thread t;
    public Print_Demo(Synchronize_Print targ,String s1,String s2)
   {
       target=targ;
       msg=s1;
       ThreadName=s2;
       t=new Thread(this);
       t.start();
   }
   public void run()
   {
       textArea1.append("\n"+ThreadName+" started executing...\n");
       target.print(msg);
       textArea1.append("\n"+ThreadName+" finished executing.\n");
    }
}
Synchronize_Print target=new Synchronize_Print();
Print_Demo ob1=new Print_Demo(target,"1","Thread-1");
Print_Demo ob2=new Print_Demo(target,"2","Thread-2");
Print_Demo ob3=new Print_Demo(target,"3","Thread-3");
// waiting threads to finish
try
{
    ob1.t.join();
    ob2.t.join();
    ob3.t.join();
}
catch(InterruptedException e)
{
    textArea1.append("Interrupted");
}
```

프로그람실행결과는 그림 3-7에서 보여준다.

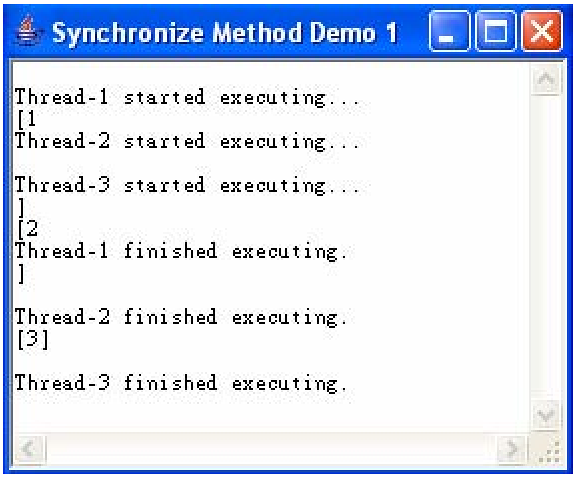


그림 3-7. 동기메쏘드를 사용할 때의 프로그람실행결과

만일 동기메쏘드를 사용하지 않으려면 void print앞의 synchronized를 없애면 된다. 즉

```
class Synchronize_print{
void print (String msg) {
```

로 설정하면 된다.

프로그람실행결과는 그림 3-8에서 보여준다.

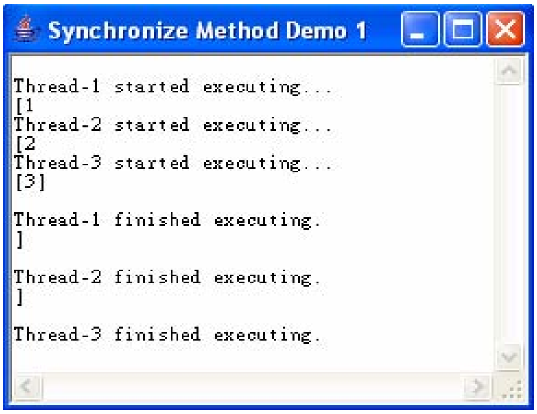


그림 3-8. 동기메쏘드를 사용하지 않은 경우의
프로그람실행결과

클라스 Synchronize_print의 print메쏘드에서 먼저 《[》와 괄호의 내용을 인쇄하고 다음에 sleep()메쏘드를 리용하여 이 스레드가 1초 잠자기한 후 다시 괄호 《]》를 인쇄한다. 동기메쏘드를 리용하는 경우 두번째와 세번째스레드가 이미 일정짜기에 의하여 실행되고 있어도 이것들은 앞의 스레드가 print()메쏘드의 전체 인쇄작업을 집행완성하기만을 기다렸다가 완성된 다음 실행에 진입할수 있다. 이것은 바로 print()메쏘드에 접근하는 동기를 실현하고있다는것을 보여준다. 만일 메쏘드 print()를 동기화하지 않으면 첫번째 스레드가 print()메쏘드에 접근하여 집행을 결속하지 못하였음에도 불구하고 두번째, 세번째 스레드는 print()메쏘드에 접근하여 실행을 하게 된다. 최종적으로는 세번째 스레드가 인쇄를 전부 완성한 다음에 첫번째와 두번째 스레드가 뒤따라 print()메쏘드를 호출하여 나머지 괄호 《]》을 인쇄한다.

3.4.3. 동기명령문

창조하려는 클라스의 내부에서 동기메쏘드를 창조하는것이 동기화를 실현하기 위한 간단한 방법이라고 하여도 그것은 아무때나 다 유효한것이 아니다. 가령 어떤 클라스가 synchronized메쏘드를 리용하지 않았다고 하자. 그리고 이 클라스는 제3의 방식으로 창조되여 그의 원천코드를 얻을수 없다고 하자. 그러면 이 메쏘드앞에 synchronized수식부를 붙일수 없으며 이때에는 다른 방법을 리용하여야 한다. 즉 동기명령문을 사용하여야 한다. 클라스에서 동기를 요구하는 코드부분을 synchronized블로크안에 넣기만 하면 된다. 형식은 다음과 같다.

```
synchronized(object){
// 동기를 요구하는 코드
}
```

여기서 object는 동기객체에 대한 인용이다. 만일 동기화하려는것이 한개의 명령문뿐이면 괄호를 붙힐 필요가 없다. 동기블로크가 object성원메쏘드에 대한 호출을 할수 있는것은 오직 현재의 스레드가 object처리관리에 성공적으로 들어선 다음이다. 앞의 실례를 가지고 설명하자. 프로그람에서 약간한 수정을 진행한다. 즉 메쏘드 void print(String msg)앞에 synchronized열쇠어를 붙이지 않는다. 그러나 run()메쏘드는 동기명령문을 사용하고있으며 결과는 동기메쏘드의 사용과 완전히 일치한다.

상응한 프로그람은 다음과 같다.

```
class Synchronize_Print
{
    void print(String msg)
    {
```

```
        textArea1.append(" ["+msg);
        try{
            Thread.sleep(1000); //sleep 1 second
        }
        catch(InterruptedException e){
            textArea1.append("Interrupted");
        }
        textArea1.append("] \n");
    }
}
class Print_Demo implements Runnable
{
    String msg;
    String ThreadName;
    Synchronize_Print target;
    Thread t;
    public Print_Demo(Synchronize_Print targ,String s1,String s2)
    {
        target=targ;
        msg=s1;
        ThreadName=s2;
        t=new Thread(this);
        t.start();
    }
    public void run()
    {
        textArea1.append("\n"+ThreadName+" started executing...\n");
        synchronized(target)
        {
           target.print(msg);
        }

        textArea1.append("\n"+ThreadName+" finished executing.\n");
    }
}
Synchronize_Print target=new Synchronize_Print();
```

```
Print_Demo ob1=new Print_Demo(target,"1","Thread-1");
Print_Demo ob2=new Print_Demo(target,"2","Thread-2");
Print_Demo ob3=new Print_Demo(target,"3","Thread-3");
// waiting threads to finish
try
{
    ob1.t.join();
    ob2.t.join();
    ob3.t.join();
}
catch(InterruptedException e)
{
    textArea1.append("Interrupted");
}
```

제5절. 스레드의 통신

Synchronized잠구기기구는 스레드사이의 호상영향을 피할수 있게 한다. 그러나 실천에서는 스레드사이의 통신방법을 요구한다. 스레드사이의 통신기구는 많다. 례하면 PipedInputStream과 PipedOutputStream를 사용하여 두 스레드사이에 바이트단위로 자료를 전송하거나 자료공유를 통하여 하나의 스레드는 자료항목을 준비하고 다른 하나의 스레드는 통보를 받아 이 자료항목을 얻은 다음 wait()와 notify()를 사용하여 통신을 진행한다. 이 내용들은 꼭 알고있어야 할 내용이므로 여기서 구체적으로 설명한다.

3.5.1. 자료공유

스레드사이에 통신하는 일반적인 방법은 하나의 스레드는 자료항목을 준비하고 다른 하나의 스레드는 통지를 받아 이 자료항목을 얻어 사용하는 방법이다.

자료항이 간단한 옹근수라면 우선 스레드를 전송하는 Runnable클라스에서는 int성원변수를 정의해야 한다. 그리고 스레드를 접수하는 Runnable클라스는 이 수값을 쓰기하고 읽는다. 그러나 우의 과정만을 거치면 수신측이 변수의 값을 정확히 받아들였는가는 알수 없다.

이 문제는 어떤 표지변수를 리용하여 실현할수 있다. 보통 boolean값을 사용한다. 송신측에서는 변수에 수값을 넣고 표지변수를 true로 설정한다. 그리고 수신측에서는 자료를 읽은 다음 표지변수를 false로 설정한다. 이것은 바로 송신측이 수신측에서 수값을 정확히 얻었는가를 알게 하며 변수에 이 수값을 쓸수 있게 한다.

실례 3-4

자료를 리용하여 스레드사이의 통신을 실현한다. 작성단계는 실례 3-1과 같다. 상응한 위치에 다음의 코드를 추가한다.

```
class FlagSend implements Runnable
{
    volatile int theValue;
    volatile boolean isValid;
    public void run()
    {
        for(int i=0;i<5;i++)
        {
            while(isValid){
                Thread.yield();
            }
            theValue=(int)(Math.random()*100);
            textArea1.append("Sending: "+theValue+"\n");
            isValid=true;
        }
    }
}
class FlagRec implements Runnable{
    FlagSend theSender;
    public FlagRec(FlagSend sender)
    {
        theSender=sender;
    }

    public void run()
    {
        while(true)
        {
            while(!theSender.isValid){
                Thread.yield();
            }
            textArea1.append("Received: "+theSender.theValue+"\n");
```

```
            theSender.isValid=false;
        }
    }
}
FlagSend s=new FlagSend();
FlagRec r=new FlagRec(s);
Thread st=new Thread(s);
Thread rt=new Thread(r);
rt.setDaemon(true);
st.start();
rt.start();
try{
    st.join();
    while(s.isValid){
        Thread.sleep(100);
    }
}
catch(InterruptedException e){
    textArea1.append("InterruptedException caught");
}
```

프로그람의 실행결과는 그림 3-9와 같다.

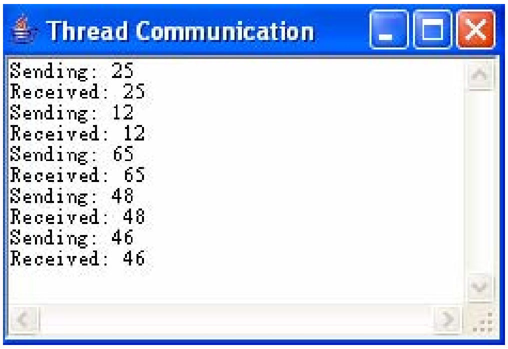


그림 3-9. 스레드사이에 공유자료를 리용하여 통신을 실현

프로그람에서 int theValue를 volatile로 수식하였다. 이렇게 선언하면 자료의 공유를 진행할수 있다. 한개의 스레드가 theValue를 사용할 때 다른 변수들도 그에 대해 수정을 할수 있다.

두개의 공유변수 theValue와 isValid는 FlagSend클라스에서 FlagSend의 객체를 생성할 때 각각 령(0)과 false로 초기화된다. 송신측이 수값을 전송할 때 그 객체는 isValid의 수값이 false로 있을 때까지 줄곧 기다린다. 송신이 시작되면 수값은 즉시 쓰기된다. 일단 수값이 쓰기되면 표지변수 isValid는 true로 설정된다. 다음에 송신측은 다음 수값을 발송한다. isValid가 true이므로 송신측은 스레드를 접수하여 이 변수를 변경시킬 때까지 기다린다.

스레드를 접수하여 기동할 때 isValid는 초기에 false이다. 이것은 스레드를 전송하여 이 변수값을 true로 바꿀 때까지 기한다. 앞에서 서술한바와 같이 isValid가 true로 되는 경우는 변수 theValue가 정확한 값으로 될 때이다. 이렇게 스레드를 접수하여 theValue를 읽어들이고 이것을 쓰기출력한다. 이 값을 읽어들인후 스레드를 접수하여 isValid를 false로 설정하고 스레드를 발송하며 다음 수값을 theValue에 넣을수 있다고 알린다.

---

isValid값은 이 프로그람설계의 정확성을 보장하는데서 관건적인 작용을 한다. 이것은 하나의 문으로 볼수 있다. 또한 송신측이 자료를 쓰기하고 아직 사용하지 않은 이전 수값들이 파괴되지 않게 한다. 그리고 수신측이 수값을 읽을수 있게 하고 낡은 수값들을 취할수 없게 한다.

이 방법의 부족점은 효률이 낮은것이다. 두개의 스레드를 고려하는것은 통신에서 보기드문것이다. 실례로 스레드를 발송할 때 오랜시간 자료를 준비하여야 하거나 수신측이 오랜시간 이 수값을 처리해야 할 때 한개의 스레드는 부단히 순환하면서 마지막이 변할 때까지 isValid값을 여러번 검측하여야 한다.

그밖에 Math.random()을 리용하여 발송수를 생성하므로 매번의 실행결과는 완전히 같지 않지만 발송과 접수통신과정은 같다.

3.5.2. wait()와 notify()를 사용한 통신실현

wait()메쏘드는 스레드가 어떤 조건을 만족할 때까지 기다리게 하며 notify메쏘드는 대기자가 이미 어떤 사건을 발생시켰다는것을 통지하여 스레드가 다시 실행상태에 들어가게 한다.

wait()와 notify()메쏘드는 Object클라스에서 정의되며 모든 클라스로부터 계승된다.

wait()메쏘드의 일반형식은 다음과 같다.

```
synchronized void MethodWait(){
while (condition)wait();
// 조건이 참일 때 아래의 명령문을 집행
}
```

• 모든 일감은 동기메쏘드안에서 집행된다. 그렇지 않으면 객체내용이 불안정하다. 실레로 메쏘드가 동기메쏘드가 아니면 다른 스레드가 while명령문의 조건을 수정할수 있기때문에 그 조건이 정확한 조건이라고 담보할수 없다.

• 조건검사는 시종일관 하나의 순환안에 있어야 한다. 항상 조건이 이미 만족한다고 가정하지 말아야 한다. 바꾸어 말하면 while을 if로 고치지 말아야 한다.

notify메쏘드는 자료를 수정하는 메쏘드들에 의하여 호출되며 다른 스레드는 이 자료들의 변동을 기다린다. 일반형식은 다음과 같다.

```
synchronized void MethodNotify() {
// 조건변동
notify();
}
```

여러 스레드는 동일한 객체를 기다린다. notify를 리용하여 기다림시간이 제일 긴 스레드를 깨운다. 만일 기다림스레드들을 모두 깨우려면 notifyAll()을 사용하면 된다. wait(), notify()와 notifyAll()메쏘드들은 프로쎄스들사이의 통신기구를 실현한다. 이 메쏘드들은 객체에서 final메쏘드를 리용하여 실현한다. 그러므로 클라스들은 모두 final메쏘드들을 포함한다. 이 3개의 메쏘드들은 동기메쏘드에서만 호출할수 있다.

아래의 실례는 요소의 삽입, 얻기, 삭제를 실현하고있는 프로그람이다.

실례 3-5

스레드사이의 통신단계는 기본적으로 실례 3-1과 같다. 상응한 위치에 아래의 코드를 추가한다.

```
class Element{
    int data;
    Element next;
    public Element(int data)
    {
        this.data=data;
        next=null;
    }
    }
```

```
class Queue{
    Element head,tail;
    boolean valueSet=false;
    public synchronized void add_element(Element p){
        if(valueSet)
            try{
                wait();
            }
        catch(InterruptedException e){
            textArea1.append("InterruptedException caught");
        }
    valueSet=true;
    if(tail==null) head=p;
    else tail.next=p;
    p.next=null;
    tail=p;
    textArea1.append("add: "+p.data+"\n");
    notify();
    }

    public synchronized void get_element(){
        if(!valueSet)
            try{
                wait();

            }
        catch(InterruptedException e){
            textArea1.append("InterruptedException caught");
        }
    Element p=head;
    head=head.next;
    if(head==null) tail=null;
    textArea1.append("get: "+p.data+"\n");
    valueSet=false;
    notify();
```

```
        }
    }

    class Producer implements Runnable{
        Queue q;

        Producer(Queue q){
            this.q=q;
            new Thread(this,"Producer").start();
        }
        public void run(){
            int i=0;
            while(i<10){
                Element e=new Element(i++);
                q.add_element(e);
            }
        }
    }

    class Consumer implements Runnable{
        Queue q;
        Consumer(Queue q){
            this.q=q;
            new Thread(this,"Consumer").start();
        }
        public void run(){
            while(true){
                q.get_element();
            }
        }
    }

    Queue q=new Queue();
    new Producer(q);
    new Consumer(q);
```

프로그람실행결과는 그림 3-10과 같다.

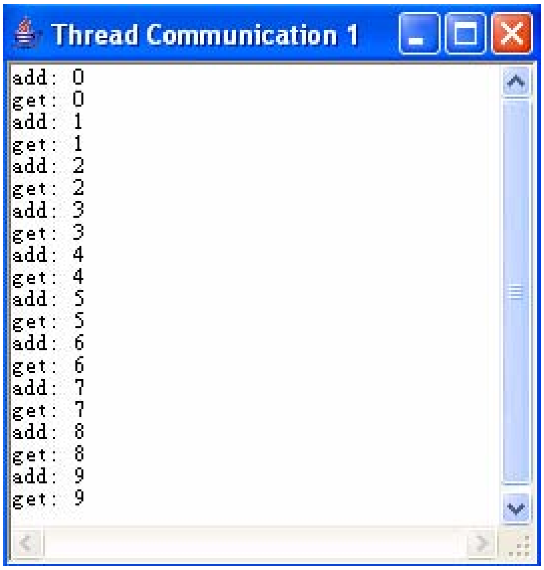


그림 3-10. 스레드들사이의 통신

결과에서 보는바와 같이 wait()와 notify()메쏘드를 리용하여 스레드사이의 통신을 실행하며 추가와 얻기 메쏘드는 동기를 실현하고있다. 이것은 단일스레드체계에서의 대기렬과 매우 류사하다. 이 프로그람에서는 wait()와 notify()메쏘드를 사용하여 요소를 대기렬에 추가하고 대기자에게 통지한다. 대기렬이 비였을 때에는 다른 스레드를 기다렸다가 내용을 삽입하므로 get메쏘드는 null을 귀환하지 않는다. 대부분 스레드들은 대기렬뒤에 한개이상의 요소를 추가할수도 있고 또한 대기렬로부터 한개이상의 요소를 얻을수도 있다.

제4장. 콤퓨터망프로그람작성

Java는 콤퓨터망을 지원하는 능력이 상당히 높다. 이 장에서는 콤퓨터망의 개념에 대하여 간단히 설명하고 망통신규약, 소케트, 통신포구, IP주소의 원리에 대하여 소개한다. 그리고 InetAddress클라스 , TCP/IP소케트, UDP데이터그램, FTP응용, Internet응용 등에 대하여서도 서술한다.

제1절. 몇가지 개념

콤퓨터망은 콤퓨터체계, 자료통신, 망프로그람의 세부분으로 구성되여있다. 콤퓨터체계는 망의 기초를 이루며 각종 망자원(하드웨어설비자원, 쏘프트웨어자원)들을 제공한다. 자료통신은 콤퓨터체계를 련결해주는 다리로서 각종 련결기술과 정보교환기술을 제공한다. 자주 쓰이는 정보교환방법에는 선로교환, 전보문교환, 그룹교환 방식이 있다. 망프로그람은 망관리자로서 각종 망봉사를 실현한다. 망프로그람에는 망규약프로그람(례를 들어 TCP/IP, IPX 등 규약프로그람)과 망봉사프로그람, 망관리프로그람, 다양한 망도구프로그람, 그리고 전문적으로 개발된 국부망조작체계(가장 대표적인 국부망조작체계로는 Netware, NT를 들수 있다.)가 포함된다.

4.1.1. 통신규약

망의 층구조와 망규약은 콤퓨터망을 구성하는데서 기본이라고 할수 있다.(그림 4-1)

규약은 어떤 콤퓨터망에서의 대화를 위한 약속들을 말하는데 실례로 n번째층과 다른 콤퓨터의 n번째층사이에 진행되는 대화에서 리용하는 일련의 규칙과 규약모임을 간단히 n번째층규약(protocol)이라고 부른다. 망의 구조해석과 설계에 대한 일반적리해를 위해 그림 4-2에서 국제표준화기구(ISO)의 OSI참조모형을 제시한다.

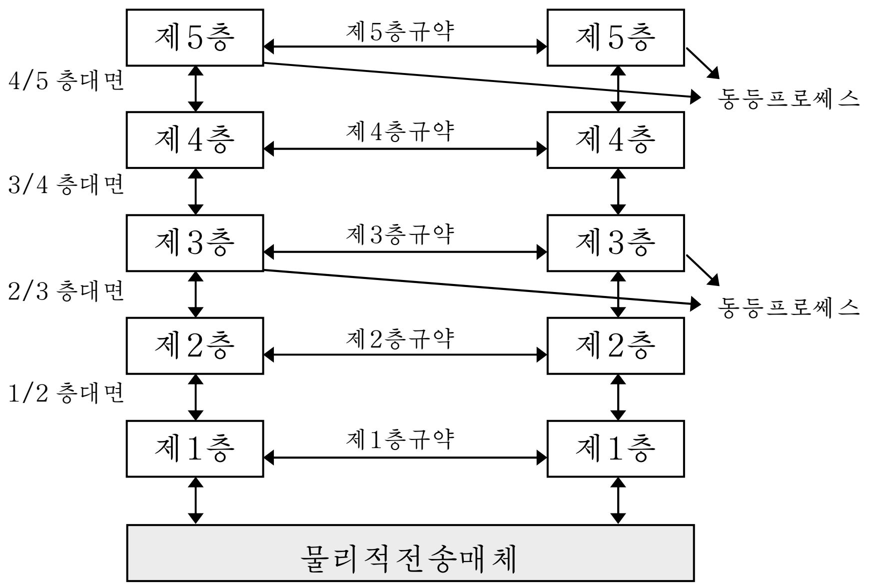


그림 4-1. 계층, 규약, 대면 및 동등프로세스의 구조

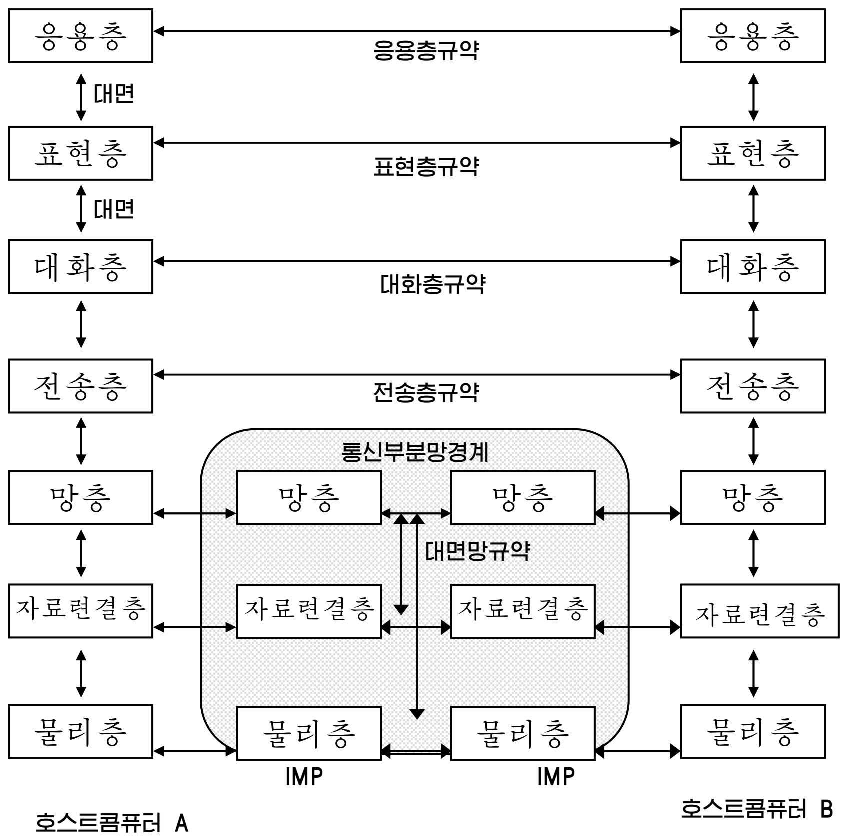


그림 4-2. 국제표준화기구의 OSI 참조모형

국제표준화기구의 OSI참조모형은 단지 망규약에 관한 참조모형일뿐이다. 인터네트나 인트라네트가 리용하는 규약들중 많이 쓰이는 규약은 TCP/IP규약이다. TCP/IP규약은 표준화된 규약묶음으로서 1969년에 개발되였으며 광역망(WAN)을 포함하는 대형인터네트를 위하여 작성된것이다.

표 4-1. TCP/IP 모형과 OSI 모형의 대응관계

OSI참조모형		TCP/IP참조모형	
7	응용층	4	응용층
6	표현층	3	전송층
5	대화층	2	망층
4	전송층	1	통신부분망층
3	망층		
2	자료련결층		
1	물리층		

TCP/IP의 목적은 고속망통신을 실현하는것이다. 이것으로 하여 1969년부터 지금까지 인터네트와 같은 세계적범위의 통신망으로 발전하게 되였다. 그림 4-3은 TCP/IP의 계층구조를 보여준다.

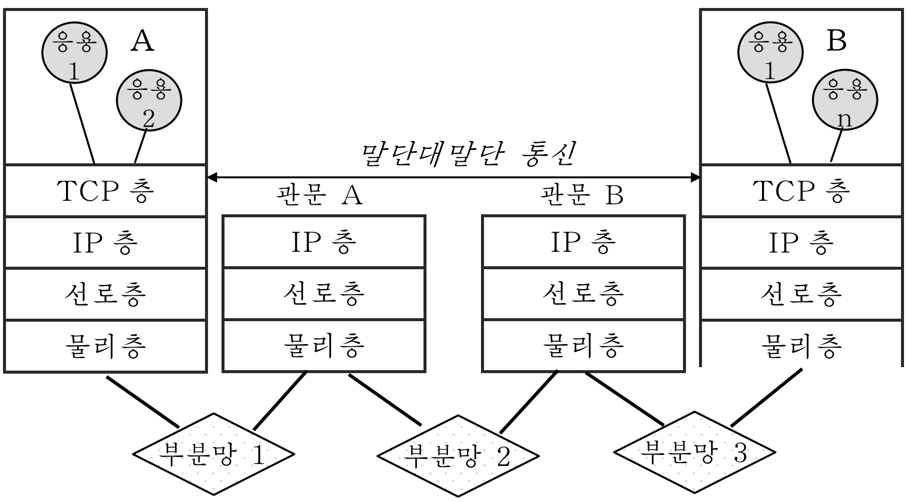


그림 4-3. TCP/IP 의 계층구조

이 그림과 표 4-1에 제시한 TCP/IP모형과 OSI모형의 대응관계를 고찰하여 보면 TCP/IP모형과 OSI모형의 관계와 차이를 알수 있다. TCP/IP모형은 OSI모형에 비해볼 때 상대적으로 간단하다. 4개의 층으로 되여있지만 실용성이 높은것으로 하여 광범히 리용되고있다.

그림 4-4에 제시한 TCP/IP망구조도는 TCP/IP망의 모든 규약에 대하여 구체적으로 보여준다. IP는 저준위경로선택규약으로서 자료를 작은 파케트로 가르고 망을 통하여 어떤 주소에 보낸다. 그러나 전송하려는 정보파케트를 반드시 목적지에 보낸다는 담보는 없다. 전송조종규약(TCP)은 비교적 높은 준위의 규약으로서 자료파케트를 하나로 묶어 필요할 때마다 배렬하고 반복전송하여 자료전송의 믿음성을 높여준다.

세번째 규약은 사용자데이터그램규약(UDP)으로서 TCP규약과 비슷하다. UDP는 비접속방법으로서 빠른 속도로 자료를 전송하는 반면에 믿음성이 높지 못하다.

응용층은 응용프로그람이 다른 층에 대한 봉사를 진행할수 있게 하는 층으로서 자료교환을 진행하는 규약층이다. 지금까지 많은 응용층규약들이 개발되였으며 새로운 규약들이 끊임없이 개발되고있다. 가장 널리 알려진 응용프로그람규약들은 아래와 같은 규약들이다.

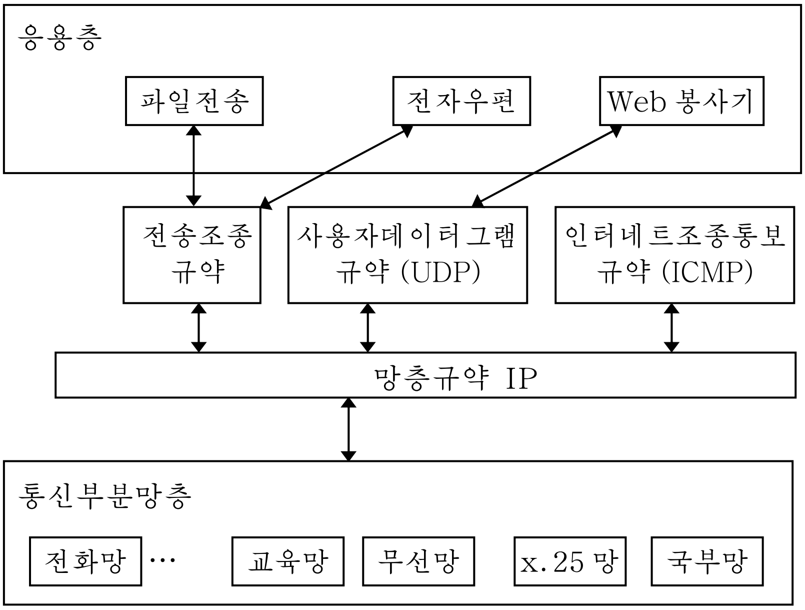


그림 4-4. TCP/IP 망구조도

- 하이퍼본문전송규약(HTTP): WWW망의 웨브페지파일을 전송하는데 리용하는 규약
- 파일전송규약(FTP): 대화식파일전송에 리용한다.
- 간단한 우편전송규약(SMTP): 우편전송과 접속에 쓴다.
- Telnet-말단모의규약: 망호스트콤퓨터에 원격으로 접속하는데 리용한다.

그밖에 아래의 응용층규약들은 TCP/IP망의 사용과 관리를 편리하게 할수 있도록 한다.

- 령역이름체계(DNS): 콤퓨터의 이름을 IP주소로 해석해준다.
- 경로선택정보규약(RIP): 일종의 경로선택규약으로서 경로기는 이 규약을 리용하여 IP망에서 경로선택정보를 교환한다.
- 간단한 망관리규약(SNMP): 망관리조작탁과 망설비(경로기, 망다리, 지능하브)들 사이에 망관리정보를 선택하고 교환하는데 리용한다.

4.1.2. 통신포구

소케트모형에서는 소케트를 리용하여 한 대의 콤퓨터가 여러개의 서로 다른 의뢰기에 봉사할수 있고 서로 다른 류형의 정보봉사를 제공할수 있다. 이것은 일반적으로 통신포구(port)를 통하여 처리된다. 통신포구는 콤퓨터번호에 대한 소케트이다. 봉사기프로쎄스는 《감시》포구를 통하여 항상 의뢰기가 그에 련결되여있는가를 감시한다. 여러개의 의뢰기접속을 관리하기 위하여서는 봉사기프로쎄스가 반드시 다중스레드이든가 동기적인 입출력처리를 할수 있어야 한다.

일단 접속에 성공하면 어떤 규약에 따라 봉사가 진행된다. 이때 규약은 사용하는 포구와 관련이 있다. TCP/IP는 구체적인 규약에 대응하기 위하여 1024개의 포구를 가지고있다. 여기서 포구 21은 FTP의 포구이고 23은 Telnet의 포구이다. 그리고 25는 전자우편의 포구이고 79는 finger포구이다. 80은 HTTP포구이다.

아래에서는 HTTP규약을 가지고 매개 규약이 통신포구를 통해 의뢰기와 대화처리하는 과정을 설명한다.

HTTP는 망열람기와 봉사기가 하이퍼본문망페지와 화상을 전송할 때 리용하는 규약이다. 이것은 기본 망페지에서 봉사기를 통하여 자료를 열람할수 있도록 하는 가장 간단한 규약이다. HTTP규약에 의한 Web열람과정을 그림 4-5에서 보여준다. 여기에서는 의뢰기가 한개 파일 index.html을 요청하고 봉사기는 이 파일을 찾아 의뢰기에 보내주는 내용을 보여주고있다.

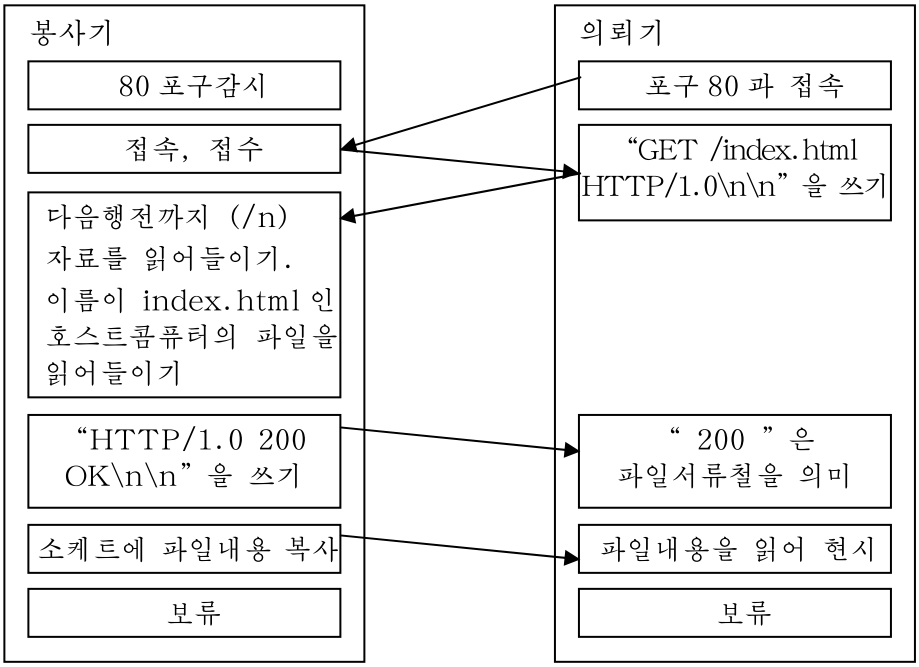


그림 4-5. HTTP 규약실현의 실례

4.1.3. IP주소

IP는 비접속이면서 믿음성이 없는 데이터그램규약으로서 기본적으로 주소를 찾고 호스트콤퓨터들사이에 파케트를 전송하기 위한 경로를 선택할 때 리용한다. 즉 IP주소는 인터네트상에서의 매 콤퓨터가 가지고있는 유일한 하나의 주소라고 말할수 있는데 그것은 망상에서 매 콤퓨터를 가리키는 《신분증》과 같다. 주의하여야 할것은 모든 TCP/IP호스트콤퓨터의 IP주소는 론리적인 IP주소를 의미하며 망층상에서의 주소로서 자료련결층에서의 주소와 관련이 없다는것이다. 호스트콤퓨터와 TCP/IP를 리용하여 통신하는 망부품들은 한개의 유일한 IP주소를 요구한다.

IP주소는 망에 존재하는 체계의 주소를 의미하는데 마치 어느 아빠트가 도시의 어느 위치에 있는가를 나타내는 거리주소와 같다.

매개 IP주소는 망식별자(ID)와 호스트콤퓨터식별자를 포괄하고있다.

- 망식별자(망주소라고도 함)는 경로기에 의하여 정해진 동일한 물리적망의 체계를 가리킨다. 이 물리적망상의 모든 체계는 반드시 꼭같은 망식별자를 가져야 한다. 망식별자는 인터네트에서 반드시 유일하여야 한다.
- 호스트콤퓨터식별자(호스트콤퓨터주소라고도 함)는 망에서의 작업장, 봉사기, 경로기, 기타 TCP/IP호스트콤퓨터를 의미한다. 호스트콤퓨터주소는 IP주소에 대하여 유일하여야 한다.

IP주소는 32bit인데 보통 8bit씩 묶는다. 매개 8bit묶음을 0~255사이의 10진수(10진수체계)로 바꾸고 점으로 구별한다. 실례로 2진수의 IP주소《11000000 10101000 00000001 00000011》를 점으로 구별된 10진수의 IP주소로 바꾸면 《192.168.1.3》으로 된다. 기억의 편리성을 위하여 일반적으로 점으로 구별한 10진법IP주소표기법을 사용한다.

인터네트조직기구는 5개의 주소클라스를 정의하여 서로 다른 크기의 망을 받아들이고있다. TCP/IP는 호스트콤퓨터에 분배하는 A,B,C,D,E클라스의 주소를 지원하고있다. 주소클라스들은 IP주소에서 어느것이 망식별자에 해당하고 어느것이 호스트콤퓨터식별자에 해당하는가를 정의한다. 동시에 가능한 망의 개수와 매개 망에서의 호스트콤퓨터개수를 정의하고있다.

- A클라스

A클라스주소는 많은 량의 호스트콤퓨터들을 가지고있는 망에 적용한다. A클라스주소의 최고 상위비트는 0으로 설정한다. 아래의 7개비트는 망식별자를 의미한다. 나머지 24개비트(3개의 8비트묶음)는 호스트콤퓨터식별자를 나타낸다. 이것은 126개의 망을 허용하며 매개 망은 16777214대의 호스트콤퓨터를 가질수 있다. 그림 4-6에서 A클라스주소의 구조를 보여주고있다.

- B클라스

B클라스주소는 중규모망에 적용한다. B클라스주소의 최고 2개비트는 2진수로 10이다. 다음의 14개비트(2개의 8비트묶음)는 망식별자를 의미한다. 나머지 16개비트(2개의 8개비트묶음)는 호스트콤퓨터식별자를 의미한다. 이것은 16384개의 망을 허용하며 매개 망은 65534대의 호스트콤퓨터를 포함할수 있다.(그림 4-6)

- C클라스

클라스주소는 소형망에 적용한다. C클라스주소의 최고 3개비트는 2진수로 110이다. 다음의 21개비트는 망식별자를 의미한다. 나머지 8개비트는 호스트콤퓨터식별자를 의미한다. 이것은 2097152개의 망을 허용하며 매개 망은 254대의 호스트콤퓨터를 포함할수 있다.(그림 4-6)

- D클라스

D클라스주소는 예비적으로 쓰는 IP주소이다. D클라스주소의 최고 4개비트는 2진수로 1110이다. 나머지 비트는 호스트콤퓨터조직자가 사용한다. Microsoft는 D클라스주소를 지원한다. 많은 경우 이 클라스의 IP주소들은 다중방송을 위한 대역으로 쓰인다.

- E클라스

E클라스주소는 하나의 실험용주소로서 예비적으로 남겨 앞으로 필요한 경우에 사용한다. E클라스주소의 최고 4개비트는 1111이다.

그림 4-7에서는 A, B, C 클라스의 IP주소범위와 사용할수 있는 망의 개수, 호스트콤퓨터개수들을 대비적으로 보여주고있다. 국부망에서는 일반적으로 C클라스주소를 사용하고 있다.

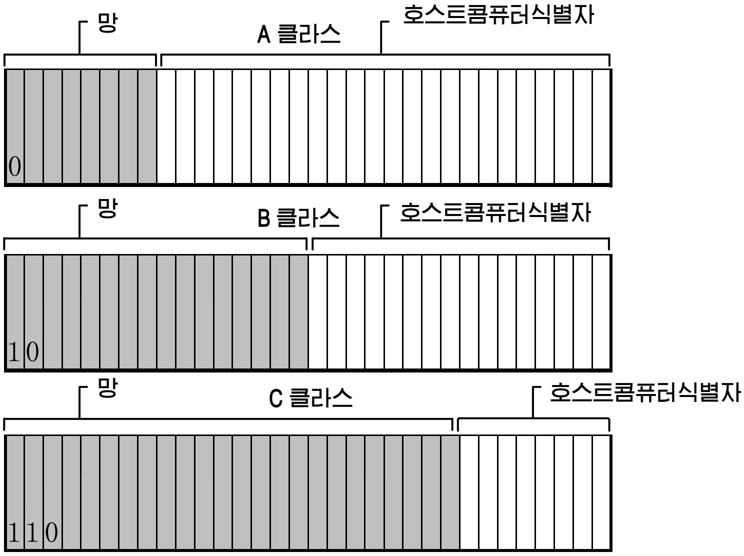


그림 4-6. IP 주소표(A, B, C 클라스)

표 4-2 IP 주소표대비

부류	W의 값	망식별자부분	호스트콤퓨터 식별자부분	사용가능한 망 수	매개 망의 호스트콤퓨터수
A	1~126	w	x.y.z	126	16777214
B	128~191	w.x	y.z	16384	65534
C	192~223	w.x.y	z	2097152	254

제2절. IP주소의 얻기

Java.net.InetAddress클라스는 Java에서 IP주소를 정의할 때 리용하는 클라스이다.

InetAddress클라스는 일반적으로 다른 망관련클라스들과 같이 사용된다. 실례로 Socket, ServerSocket, URL, DatagramSocket, DatagramPacket클라스 등과 함께 사용된다.

InetAddress클라스는 Object로부터 직접 파생된 클라스로서 아래와 같이 정의된다.

public final class InetAddress extends Object implements Serializable

이 클라스는 인터네트주소를 표시하는 2개의 마당 즉 hostName(String)과 address(int)를 포함하고있다. hostName은 콤퓨터의 이름을 말한다. 례를 들어 www.sec.com.kp.로 표현된다. address는 32bit의 IP주소를 의미한다. 이 마당들은 공개마당이 아니므로 직접 접근할수 없다.

표 4-3에 InetAddress클라스에서 자주 리용하는 메쏘드와 그에 대한 설명을 주었다.

표 4-3. InetAddress클라스의 메쏘드들

메 쏘 드	의 미
boolean equals(other) (Object other)	만일 객체가 other와 같은 인터네트주소를 가지면 true를 귀환한다.
byte[] getAddress()	객체의 인터네트주소를 4개의 바이트원소들을 가진 배렬을 얻는다.
String getHostAddress()	InetAddress객체의 호스트콤퓨터주소를 문자렬로 귀환한다.
String getHostName()	InetAddress객체의 호스트콤퓨터이름을 문자렬로 귀환한다.
Int hashCode()	객체를 호출하는데 리용하는 하쉬표를 귀환한다.
boolean isMulticast Address()	인터네트주소가 다중방송주소이면 true를 귀환하고 그렇지 않으면 false를 귀환한다.
String toString()	호스트콤퓨터이름문자렬과 IP주소를 귀환한다.

InetAddress클라스에서 가장 많이 쓰는 메쏘드는 getByName()이다. 이 메쏘드는 정적메쏘드로서 찾으려는 콤퓨터의 이름을 파라메터로 요구하며 령역이름체계(DNS)를 리용하여 IP 주소를 찾는다. getByName()메쏘드의 형식은 다음과 같다.

java.net.InetAddress address=
=java.net.InetAddress.getByName(“www.sec.com.kp”);

만일 제시된 콤퓨터를 찾지 못하면 이 메쏘드는 UnknownHostException례외 혹은 그의 상위례외인 IOException을 발생시킨다. 이 례외처리방법은 아래와 같다.

```
try{
   InetAddress address=InetAddress.getByName( “www.sec.com.kp” );
   System.out.println(address);
}
catch(UnknownHostException e){
   System.out.println( “Could not find www.sec.com.kp” );
}
```

InetAddress클라스에는 그밖에 getLocalHost()메쏘드가 있는데 이 메쏘드 역시 정적메쏘드로서 호스트콤퓨터의 IP주소를 얻는 매우 중요한 메쏘드이다.

InetAddress.getByName()메쏘드처럼 콤퓨터의 주소를 찾지 못하였을 경우 UnknownException례외를 발생시킨다. 이 메쏘드의 형식은 다음과 같다.

InetAddress thisComputer=InetAddress.getLocalHost();

실례 4-1(호스트콤퓨터의 IP와 봉사기의 IP를 현시)

JBuilber에서 설계순서는 아래와 같다.

1. Project Wizard를 리용하여 프로젝트창조

Project Wizard를 리용하여 이름이 ***.jpx인 프로젝트를 창조한다. Project Wizard를 리용하는 구체적인 단계는 다음과 같다.

단계

1 【File】→【New Project】지령을 선택한다.
2 【Name:】칸에 《getIP》를 입력한다.
3 【Finish】 단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】→【New】지령을 선택한다.
2 Application아이콘을 두번 찰칵한다. 이때 Application Wizard대화칸이 나타난다.
3 추가선택항목들을 기정으로 선택하고 【Finish】단추를 찰칵한다.

3. 프레임의 설정과 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.
2 Design태브를 찰칵하면 대면부설계창문이 나타난다.
3 부품선택판에서 Swing서고의 jButton을 선택하여 기본창문에 추가한다.
4 Swing서고의 jLabel을 선택하여 기본창문에 추가한다. 이 조작을 두번 반복한다.
5 Swing서고의 jTextField을 선택하여 기본창문에 추가한다.

4. jTextFiled1의 text속성을 《ppp-winxp》으로, jButton1의 text를 《IP주소정보의 조사》로 고친다.

5. Frame1.java의 패키지선언부분에서 《import java.net.*》를 추가하고 변수선언부분에 아래의 코드를 입력한다.

```
InetAddress myIPaddress=null;
InetAddress myServer=null;
```

6. jButton1의 actionPerformed사건추가

```
void jButton1_actionPerformed(ActionEvent e){
   try {
       myIPaddress=InetAddress.getLocalHost();
   }
   catch (UnknownHostException e1) {
       System.out.println("get Local Host Exception:"+e1);
   }
   try {
       myServer=InetAddress.getByName(jTextField1.getText());
   }
   catch (UnknownHostException e2) {
       System.out.println("get remote Host Exception:"+e2);
   }//IP주소를 얻은 결과를 출구
   jLabel1.setText("이 주기계의 IP주소:"+ myIPaddress.getHostAddress());
   jLabel2.setText("입력한 봉사기의 IP주소:"+ myServer.getHostAddress());
}
```

7. 콤파일 및 프로그람의 실행

콤파일 및 실행순서는 다음과 같다.

1 【File】→【Save All】지령을 선택하여 프로그람을 보관한다.

2 【F9】건을 눌러 콤파일 및 실행을 진행한다.

【IP주소정보의 조사】단추를 찰칵하면 그림 4-8과 같은 결과가 얻어진다.

TCP/IP망에서 IP주소를 리용하여 그 주소의 콤퓨터이름을 찾으려는 경우 getHostName()메쏘드를 리용한다. 여기서 IP주소는 문자렬로 쓸수도 있고 바이트묶음형식으로 쓸수도 있다.

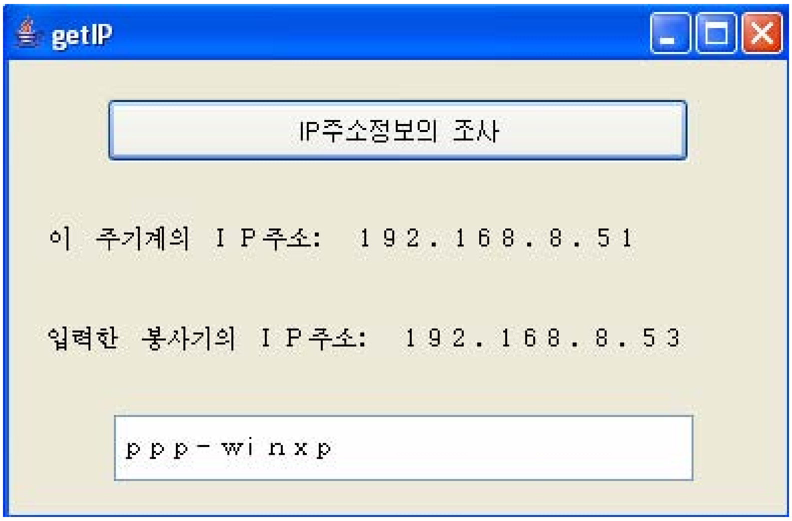


그림 4-8. IP주소찾기실례의 실행결과

InetAddress클라스에는 setHostName()메쏘드가 없다. 그러므로 java.net가 아닌 패키지들은 InetAddr객체의 내용을 변화시킬수 없다.

getHostName()은 다음과 같이 선언된다.

public String getHostName()

getHostName()메쏘드는 문자렬을 귀환하는데 이 문자렬은 InetAddress객체가 표시하는 IP주소의 콤퓨터이름이다. 만일 찾으려는 콤퓨터이름이 없거나 Applet의 보안상 특성으로하여 콤퓨터의 이름을 찾지 못하는 경우 이 메쏘드는 점으로 구분된 10진수형식의 IP주소를 귀환한다.

```
InetAddress machine=InetAddress.getLocalHost();
String localhost=machine.getHostName();
```

실례 4-2 (IP주소를 리용하여 콤퓨터이름얻기)

JBuilber에서의 설계순서는 다음과 같다.

1. Project Wizard를 리용하여 프로젝트를 창조

Project Wizard를 리용하여 이름이 ***.jpx인 프로젝트를 창조한다. Project Wizard를 리용하는 구체적인 단계는 다음과 같다.

단계

1 【File】→【New Project】지령을 선택한다.

2 【Name:】본문칸에 《getHostName》을 입력한다.

3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】→【New】지령을 선택한다.
2 Application아이콘을 두번 찰칵한다. 이때 Application Wizard대화칸이 나타난다.
3 추가선택항목들을 기정으로 선택하고 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.
2 Design태브를 찰칵하면 대면부설계창문이 나타난다.
3 부품선택판에서 Swing서고의 jButton을 선택하여 기본창문에 추가한다.
4 Swing서고의 jLabel을 선택하여 기본창문에 추가한다. 이 조작을 두번 반복한다.
5 Swing서고의 jTextField을 선택하여 기본창문에 추가한다.

4. jTextField1의 text속성을 《192.168.8.53》으로, jButton1의 text속성을 《IP주소에 의한 호스트콤퓨터이름조사》로 고친다.

5. Frame1.java의 패키지선언부분에 《import java.net.*》를 추가하고 변수선언부분에 아래의 코드를 입력한다.

```
InetAddress myServer=null;
```

6. jButton의 actionPerformed사건처리 추가

```
void jButton1_actionPerformed(ActionEvent e){
//jTextField에 입력된 봉사기 IP주소를 얻기
    try {
      myServer=InetAddress.getByName(jTextField1.getText());
    }
    catch (UnknownHostException ie) {
      System.out.println("get remote Host Exception:"+ie);
}
//IP주소를 얻은 봉사기의 호스트콤퓨터이름을 출력
jLabel1.setText("입력한 봉사기의 IP주소:"+ myServer.getHostAddress());
jLabel2.setText("봉사기의 호스트콤퓨터이름:"+ myServer.getHostName());
}
```

7. 콤파일 및 프로그람의 실행

콤파일 및 실행순서는 다음과 같다.

단계

1 【File】→【Save All】지령을 선택하여 프로그람을 보관한다.

2 【F9】건을 눌러 콤파일을 진행하고 실행시킨다.

【IP주소에 의한 호스트콤퓨터이름조사】단추를 찰칵하면 그림 4-9과 같은 결과가 얻어진다.

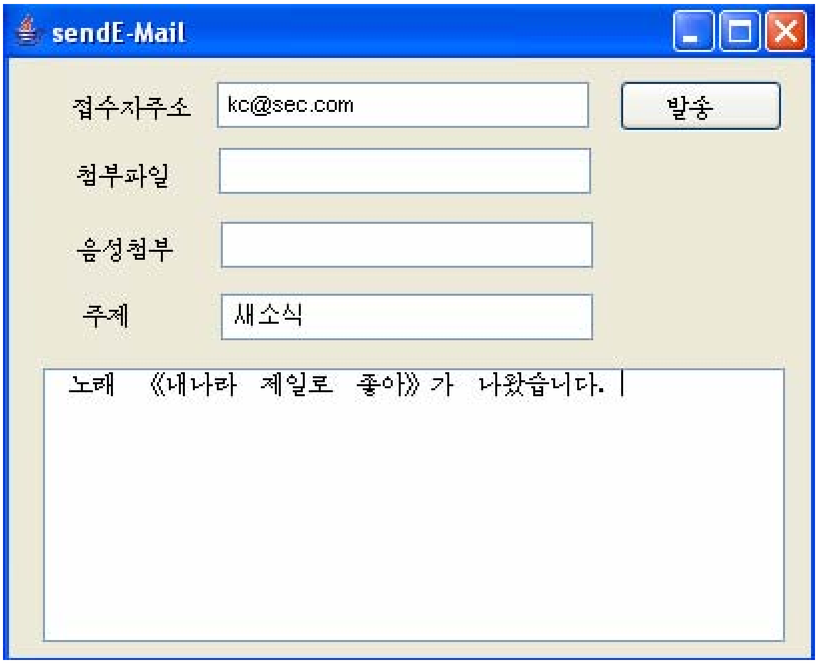


그림 4-9. IP주소를 리용하여 콤퓨터이름을 얻는 실례의 실행결과

제3절. TCP와 소케트

4.3.1. 소케트통신의 기초

Java언어에서 TCP/IP소케트접속은 java.net패키지의 클라스들을 리용하여 실현한다. Java에는 2가지 TCP소케트가 있다. 하나는 봉사기용이고 다른 하나는 의뢰기용이다. 하나의 소케트는 2개의 흐름을 가진다. 즉 한개의 입력흐름과 출력흐름을 가진다. 망을 통하여 자료를 발송할 때에는 소케트와 관련있는 출력흐름에 자료를 쓰며 자료를 받을 때에는 소케트와 관련있는 입력흐름으로부터 자료를 읽어들인다. 두 콤퓨터사이의 접속을 실현하려면 반드시 한 콤퓨터에서는 접속을 기다리고 다른 콤퓨터에서는 접속을 시도하여야 한다. 이것은 얼핏보면 전화체계와 류사하다. 이때 반드시 접속하려는 콤퓨터의 주소 혹은 이름을 알아야 한다. 그리고 매 접속에는 포구가 대응된다. 그림 4-10에서 봉사기와 의뢰기의 관계를 보여준다.

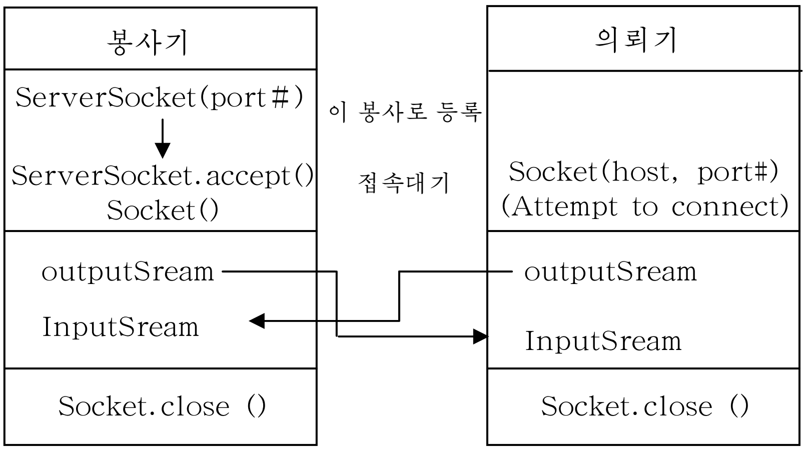


그림 4-10. 봉사기와 의뢰기사이의 접속

1 봉사기에 하나의 포구를 설정해준다. 만일 의뢰기가 접속을 시도하면 봉사기는 accept()메쏘드로 소케트접속을 진행한다.

2 의뢰기가 봉사기의 port포구를 리용하여 접속한다.

3 봉사기와 의뢰기사이의 통신은 입력흐름(Input Stream), 출력흐름(OutStream)을 리용하여 진행한다.

표 4-4에 의뢰기소케트의 몇가지 구성자들에 대하여 주었다.

표 4-4. 의뢰기소케트를 만드는 구성자들

구 성 자	의 미
Protected Socket()	하나의 접속을 만든다. 이것은 UnknownHost Exception 혹은 IOException례외를 발생시킬수 있다.
Socket(String hostname, int Port)	지정된 주콤퓨터와 지정된 포구로 접속을 진행한다. 이 메쏘드는 UnknownHostException 혹은 IOException례외를 발생시킬수 있다. 례: try{ Socket mylink = new Socket ("www.sec.com.kp",80); } catch (UnknownHostException e) { System.err.println(e); } catch(IOException e){ System.err.println(e); } 이 구성자는 콤퓨터이름을 해석할수 없거나 무의미할때 UnknownHostException례외를 발생시킨다. 기타 다른 원인으로 소케트를 열지 못하면 IOException 례외를 발생시킨다. 이 구성자는 Socket객체를 창조할뿐아니라 여러번 접속을 시도하여 어떤 포구로 접속을 할수 있는가를 판단한다.
Socket(String host, int port,boolean stream)	지정된 콤퓨터와 포구에 대한 소케트접속을 진행한다. 만일 stream변수가 true이면 흐름소케트를 창조하고 faluse이면 데이터그램소케트를 창조한다.
Socket(String host,int port, InetAddress address, Boolean stream)	지정된 콤퓨터와 포구에 대한 소케트접속을 진행한다. 이것은 UnknownHostException 혹은 IOException례외를 발생시킬수 있다. 우의 메쏘드와 차이점은 구성자에 address라는 파라메터가 하나 더 있는것이다.

Socket(InetAddress Address,int port)	지정된 콤퓨터의 포구와 접속하는 소케트를 창조한다. 접속이 실패하면 여러번 반복한다. 이 구성자는 InetAddress객체를 리용하여 접속을 진행한다. 만일 접속할수 없으면 IOException례외를 발생시킨다. 례를 들어 try{ InetAddress myaddr= InetAddress.getByName(“www.oreilly.com”); Socket mySocket=new Socket(myaddr,80); } catch(UnknownHostException e){ System.err.println(e); } catch(IOException e){ System.err.println(e); }
Socket(InetAddress address, int port,boolean stream)	지정된 InetAddress객체와 포구를 리용하여 소케트를 창조한다. 이 메쏘드는 IOException례외를 발생시킬수 있다. 만일 stream변수가 true이면 흐름소케트를 창조하고 faluse이면 데이터그램소케트를 창조한다.
Socket(String host,int port, InetAddress localAddress,int localPort)	지정된 콤퓨터의 지정된 포구에 대한 소케트를 창조한다. 앞의 두 파라메터는 접속하려는 콤퓨터와 포구이고 나머지 두개의 파라메터는 접속을 시도하는 콤퓨터와 포구이다. 망접속대면은 물리적(여러개의 망카드)인것일수도 있고 가상대면(다목적콤퓨터)일수도 있다. LocalPort를 0으로 주면 Java는 1024~65535사이의 임의의 수를 포구번호로 선택한다.
Socket(InetAddress address,int port,InetAddress localAddr,int localport)	이 구성자는 Socket(String host, int port, InetAddress localAddr,int localPort)와 비슷하나 접속하려는 호스트콤퓨터의 이름을 String형이 아니라 InetAddress로 자료를 전송한다. 이것은 TCP소케트에서 지정된 호스트콤퓨터의 지정된 포구에로의 접속을 보장한다. 또한 접속될 때까지 접속을 시도한다. 만일 실패하면 IOException례외를 발생시킨다.
Protected Socket(SocketImpl impl)	이 구성자는 SocketImpl을 리용하여 소케트를 창조한다. Java1.0이상에서만 리용할수 있다. 이 구성자는 새로운 Socket객체를 창조할 때 SocketImpl객체 impl을 설치한다. 이것은 소케트를 창조만할뿐 접속을 진행하지 않는다.

우에서 소개한 소케트구성자에서 InetAddress는 인터네트의 표준주소이고 address는 소케트통신의 지정된 IP주소이다. Port는 접속하려는 포구로서 0~65535사이의 옹근수값을 가진다. 표 4-5에서 소개한 메쏘드들을 리용하여 소케트의 주소와 그와 련관되는 포구정보를 임의의 시각에 얻을수 있다.

표 4-5. Socket 클라스의 메쏘드들

메 쏘 드	의 미
InetAddress getInetAddress()	Socket객체와 련관되는 InetAddress를 얻는다.
InetAddress getLocalAddress()	접속을 시도하는 콤퓨터의 IP주소를 얻는다.
int getPort()	소케트접속을 진행하려는 콤퓨터의 포구번호를 얻는다.
int getLocalPort()	접속을 시도하는 콤퓨터의 포구번호를 얻는다.
int getReceiveBufferSize()	수신완충기의 크기를 얻는다.
void setReceiveBufferSize()	수신완충기의 크기를 설정해준다.
int getSendBufferSize()	발신완충기의 크기를 얻는다.
void SetSendBufferSize()	발신완충기의 크기를 설정한다.

Socket객체가 일단 창조되면 그와 접속된 입출력흐름의 리용권한을 얻을수 있다. 만일 소케트가 망접속이 끊어져 실패했다면 표 4-6의 메쏘드들은 IOException례외를 발생시킨다.

표 4-6. Socket 클라스의 입출력메쏘드들

메 쏘 드	의 미
InputStream get InputStream()	소케트와 관련있는 InputStream클라스를 귀환한다. 즉 입력흐름을 얻는다.
OutputStream get OutputStream()	소케트와 관련있는 OutputStream클라스를 귀환한다. 즉 출력흐름을 얻는다.
void close()	입력흐름과 출력흐름을 닫는다. 즉 접속을 끝낸다.

Java에서는 ServerSocket클라스를 리용하여 봉사기를 창조하고 봉사기는 원격의뢰기의 공용포구를 통하여 접속을 감시한다.

ServerSocket의 구성자메쏘드의 파라메터로는 접속하려는 포구번호와 그 포구에서의 기다림시간이 리용된다. 기다림시간의 기정값은 50s이다. 구성자메쏘드는 불리한 상황이 조성되면 IOException례외를 발생시킨다. 표 4-7에 ServerSocket클라스의 몇가지 구성자들에 대하여 주었다.

표 4-7. ServerSocket 클라스의 구성자들

구 성 자	의 미
ServerSocket(int port)	지정된 포구에 기다림시간이 50s인 봉사기소케트를 창조한다.
ServerSocket(int port,int max Queue)	지정된 포구에 기다림시간의 최대값이 maxQueue인 봉사기소케트를 창조한다.
ServerSocket(int port,int maxQueue,InetAddress localAddress)	지정된 포구에 기다림시간의 최대값이 maxQueue인 봉사기의 소케트를 창조한다. 여러개의 주소를 가진 호스트콤퓨터에서 localAddress는 호스트콤퓨터가 리용하는 IP 주소이다.

ServerSocket에는 accept()메쏘드가 있는데 이 메쏘드는 의뢰기와 통신하기를 기다렸다가 의뢰기와 통신할 때 그의 Socket를 귀환한다.

4.3.2. 의뢰기/봉사기프로그람

의뢰기/봉사기응용프로그람을 작성해보자. TCP/IP봉사기응용프로그람은 Java언어가 제공하는 망클라스를 리용한다. ServerSocket클라스는 봉사기에 필요한 대부분의 작업을 진행한다.

실례 4-3(ServerSocket 클라스를 리용한 TCP/IP 의 실현)

설계순서는 다음과 같다.

1. Project Wizard를 리용한 프로젝트의 창조

단계

1 【File】→【New Project】지령을 찰칵한다.

2 【Name:】본문칸에 《TCPServer》를 입력한다.

3 【Finish】단추를 찰칵한다.

2. Application Wizard리용

단계

1 【File】 →【New】지령을 찰칵한다.

2 Application아이콘을 두번 찰칵하면 Application Wizard대화칸이 펼쳐지는데 【Title:】본문칸에 《Server Side》라고 입력한다.

3 나머지 추가선택항목들을 기정으로 설정하고 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.

2 Design태브를 찰칵한다.

3 부품선택판에서 Swing서고의 jTextArea부품을 선택하여 기본창문에 추가한다.

4 jTextField부품을 추가한다.

5 jButton부품을 추가한다.

6 jButton부품을 하나 더 추가한다.

7 jLabel부품을 추가한다.

4. 부품들의 속성값수정

단계

1 jButton1, jButton2의 text속성을 《발송시작》, 《탈퇴》로 한다.

2 jTextField1의 text속성에서 기정값을 지우고 공백으로 한다.

3 jLabel1의 text속성을 《통보:》로 수정한다.

4 jTextArea1의 text속성에서 기정값을 지우고 공백으로 한다.

5. 코드추가

추가하여야 할 코드는 아래와 같다.

```
import java.net.*;
import java.io.*;
```

우의 코드를 Framel.java의 패키지선언부분에 입력한다.
public class Framel extends jFrame에 implements Runnable를 추가한다.
다음 Framel의 변수선언부분에 아래의 코드를 입력한다.

```
ServerSocket server=null;
ServerSocket=null;
BufferedReader in=null;
PrintWriter out=null;
InetAddress myServer=null;
```

6. **프로그람코드의 작성**

단계

1 jButton1부품을 선택하고 오른쪽에 있는 사건태브를 찰칵한 다음 actionPerformed를 두번 찰칵한다. 이때 원천코드에는 jButton_actionPerfomed()메쏘드가 자동적으로 추가된다. 코드는 아래와 같다.

```
void jButton1_actionPerformed(ActionEvent e){
out.println(jTextField1.getText());
out.flush();
jTextAreal.append( “server information:” +jTextField.getText()+ “\n” );
jTextField.setText( “ ” );
}
```

2 jButton2부품에 대하여 우와 같은 방법으로 아래의 코드를 삽입한다.

```
void jButton2_actionPerformed(ActionEvent e) {
    try{
    out.println("server exit!");
    out.flush();
    }catch(Exception e2){}
    finally{
      System.exit(0);
```

```
    }
  }
```

3 jTextField1부품에 대하여 우와 같은 방법으로 사건창문의 keyPressed를 두번 찰칵하여 만들어진 jTextField1.keyPressed()메쏘드에 아래의 코드를 입력한다.

```
void jTextField1_keyPressed(KeyEvent e) {
    int j=e.getKeyCode();
    if(j==e.VK_ENTER){
    out.println(jTextField1.getText());
       out.flush();
       jTextArea1.append("server information:"+jTextField1.getText()+"\n");
       jTextField1.setText("");
     }
  }
```

4 Frame1.java에 run()메쏘드를 추가한다.

```
public void run() {
  try{
      //5438포구를 봉사기의 포구로 설정
    server = new ServerSocket(5438);
    socket = server.accept();
    in = new BufferedReader (new InputStreamReader
                                  (socket.getInput Stream()));
    out = new PrintWriter(socket.getOutputStream());
    if (socket!= null){
       jTextArea1.append("system information:  client have joined!\n");
       jButton1.setEnabled(true);
    }
    receiver r = new receiver();
    Thread t = new Thread(r);
    t.start();
  }catch(Exception e){
```

```
      jTextArea1.append(e.toString()+"\n");
    }
}
```

5 Frame1.java에 receive클라스를 추가한다.

```
private class receiver implements Runnable{
    public void run(){
      String s1 = null;
      try{
        s1 = in.readLine();
        while(s1!= "client exit!"){
          jTextArea1.append("client information:  "+s1+"\n");
          s1 = in.readLine();
        }
        in.close();
        out.close();
        socket.close();
        server.close();
      }catch(Exception e){}
        jButton1.setEnabled(false); //의뢰기측은 이미 탈퇴, 통보발송중지
    }
  }
```

6 processWindowEvent(WindowEvent e)메쏘드를 다음과 같이 수정한다.

```
protected void processWindowEvent(WindowEvent e) {
    super.processWindowEvent(e);
    if (e.getID() == WindowEvent.WINDOW_CLOSING) {
      //코드추가
      try{
        out.println("server exit!");
        out.flush();
        }catch(Exception ex){}
        finally{
```

```
            System.exit(0);
          }
       }
   }
```

7 Jbinit()메쏘드에 아래와 같은 코드를 추가

```
Thread thread=new Thread(this);
Thread.start();
```

7. 콤파일 및 실행

단계

1 【File】 →【Sava All】지령을 찰칵하여 프로젝트를 보관한다.
2 【F9】건을 눌러 콤파일 및 실행을 진행한다.

실행결과는 그림 4-11와 같다.

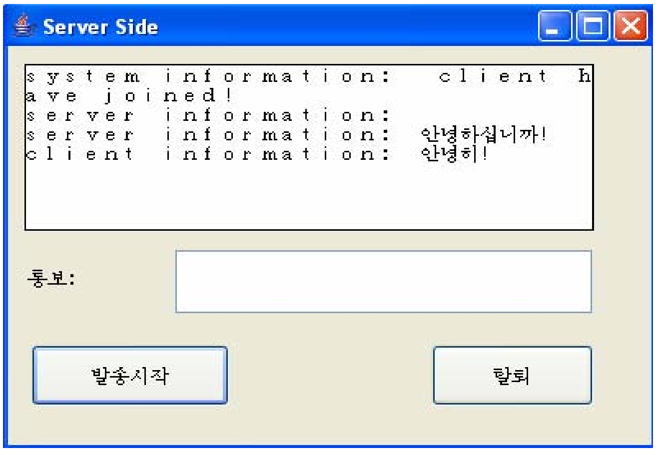


그림 4-11. TCP/IP 봉사기의 실례

실례 4-4(ServerSocket 클라스를 리용한 TCP/IP 의뢰기프로그람의 작성)

프로그람작성순서는 다음과 같다.

1. Project Wizard를 리용하여 프로젝트 창조

단계

1 【File】→【New Project】지령을 찰칵한다.

2 【Name:】본문칸에 《TCPClient》를 입력한다.

3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】→【New】지령을 찰칵한다.

2 Application아이콘을 두번 찰칵하면 Application Wizard대화칸이 펼쳐지는데 【Title:】본문칸에 《Client Side》라고 입력한다.

3 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.

2 Design태브를 찰칵한다.

3 Swing서고의 jTextArea부품을 선택하여 기본창문에 추가한다.

4 jTextField부품을 추가한다.

5 jButton부품을 3개 추가한다.

6 jLabel부품을 추가한다.

4. 부품들의 속성값수정

단계

1 jButton1, jButton2, jButton3의 text속성을 《발송시작》, 《접속》, 《탈퇴》로 수정한다.

2 jTextField1의 text속성에서 기정값을 지운다.

3 jLabel1의 text속성을 《통보:》로 입력한다.

4 jTextArea1의 text속성에서 기정값을 지운다.

5. 변수설정

단계

1 Frame1의 import구역에 아래의 코드를 삽입한다.

```
import java.net.*;
import java.io.*;
```

2 public class Frame1 extends JFrame에 implements Runnable를 추가한다.

3 Frame1에 변수들을 추가한다.

```
Socket socket=null;
BufferedReader in=null;
PrintWriter out=null;
```

6. 프로그람코드의 작성

단계

1 jButton1부품에 action Performed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton1_actionPerformed(ActionEvent e) {
    out.println(jTextField1.getText());
    out.flush();
    jTextArea1.append("client information:"+jTextField1.getText()+"\n");
    jTextField1.setText("");
}
```

2 jButton3부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton3_actionPerformed(ActionEvent e) {
  try{
  out.println("client exit!");
  out.flush();
  }catch(Exception e2){}
  finally{
    System.exit(0);
  }
}
```

3 jButton2부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton2_actionPerformed(ActionEvent e) {
  Thread thread = new Thread(this);
  thread.start();
}
```

4 jTextField1부품을 선택하고 사건태브를 찰칵한다. keyPressed사건을 두번 찰칵하면 원천코드에 자동적으로 jTextField1_keyPressed()메쏘드가 추가된다. 대응하는 코드는 다음과 같다.

```
void jTextField1_keyPressed(KeyEvent e) {
  int j=e.getKeyCode();
  if(j==e.VK_ENTER){
    out.println(jTextField1.getText());
    jTextArea1.append("client information:"+jTextField1.getText()+"\n");
    jTextField1.setText("");
  }
}
```

5 Frame1.java에서 run()메쏘드를 추가한다. 대응하는 코드는 다음과 같다.

```
public void run() {
    try{
      socket = new Socket("pio",5438);
      in = new BufferedReader(new  InputStreamReader
                                        (socket.getInputStream()));
      out = new PrintWriter(socket.getOutputStream());
      jButton1.setEnabled(true);
      receiver r = new receiver();
      Thread t = new Thread(r);
      t.start();
      jTextArea1.append("system information: have joined to server! \n");
      jButton2.setEnabled(false);
    }catch(Exception e){
      jTextArea1.append(e.toString()+"\n");
    }
  }
```

6 Frame1.java에 receive클라스를 추가한다.

```
private class receiver implements Runnable{
    public void run(){
      String s1 = null;
      try{
        s1 = in.readLine();
        while(s1!= "client exit!"){
          jTextArea1.append("client information: "+s1+"\n");
          s1 = in.readLine();
        }
        in.close();
        out.close();
        socket.close();
      }catch(Exception e){}
       jButton1.setEnabled(false);
    }
}
```

7 processWindowEvent(WindowEvent e)메쏘드를 아래와 같이 수정한다.

```
protected void processWindowEvent(WindowEvent e) {
  super.processWindowEvent(e);
  if (e.getID() == WindowEvent.WINDOW_CLOSING) {
  try{
   out.println("client exit!");
   out.flush();
   }catch(Exception ex){}
   finally{
     System.exit(0);
   }
  }
```

7. 콤파일 및 실행

단계

1 【File】→【Save All】지령을 찰칵하여 프로젝트를 보관한다.

2 【F9】건을 눌러 콤파일 및 실행을 진행한다.

결과는 그림 4-12과 같다.

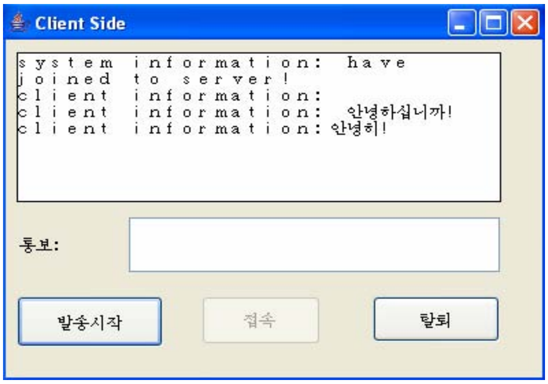


그림 4-12. TCP/IP의뢰기측 프로그람의 실례

제4절. UDP와 데이터그람

데이터그람(Datagrams)은 콤퓨터사이에 정보를 교환하는 자료형식이다. UDP통신에서는 데이터그람이 목적지에로 발송되였어도 그것이 정확히 가닿을수 있다는 담보는 없다. 심지어 수신자가 있는가에 대한 담보도 없다. 또한 수신자 역시 받은 자료가 정확한가를 담보하지 못한다. 이 절에서는 UDP통신방법에 대하여 서술한다.

4.4.1. 데이터그람통신의 기초

Java는 2개의 클라스를 리용하여 UDP규약의 데이터그람통신을 실현한다. 그중 DatagramPacket객체는 자료용기로서 전송하려는 자료를 표시하는데 리용한다.

DatagramSocket는 DatagramPacket를 발송하고 수신하는데 리용하는 통신방식을 규정하는 클라스이다.

DatagramPacket는 4개의 구성자메쏘드를 가지고있다.

표 4-8. DatagramPacket 클라스의 구성자

구 성 자	의 미
DatagramPacket(byte[]buff, int length)	길이가 length인 데이터그램를 창조
DatagramPacket(byte[]buff, int	address와 같은 주소의 콤퓨터에 발송할

length, InetAddress address)	자료길이가 length인 데이터그램을 창조
DatagramPacket(byte[]buff, int offset, int length)	길이가 length인 데이터그램을 창조한다. 이때 자료의 편위주소는 offset이다.
DatagramPacket(byte []buff, int offset, int length, InetAddress address)	address와 같은 주소의 콤퓨터에 발송할 길이가 length인 데이터그램를 창조한다. 자료의 편위주소는 offset이다.

표 4-9에서는 DatagramPacket클라스의 성원메쏘드들을 주었다.

표 4-9. DatagramPacket 클라스의 성원메쏘드

메 쏘 드	의 미
InetAddress getAddress()	InetAddress객체를 귀환한다. 보통 발송에 리용한다.
Byte[] getData()	데이터그램의 자료를 바이트묶음형식으로 귀환한다. 이 메쏘드는 받은 자료의 내용을 보는데 리용한다.
Int getOffset()	편위주소를 얻는다.
int getPort()	포구번호를 귀환한다.
int getLength()	getData()메쏘드로 얻은 자료의 길이를 귀환한다. 일반적으로 이 길이는 전체 바이트길이와 다르다.
void setData(byte[] buf)	발송하려고 하는 데이터그램자료를 바이트형식으로 설정한다.
void setData(byte[] buf, int offset, int length)	발송하려고 하는 지정된 편위주소와 길이를 가진 데이터그램자료를 설정한다.
void setLength()	바이트묶음의 유효길이를 정해준다.
void setPort()	포구번호를 설정한다.

DatagramSocket클라스의 구성자를 표 4-10에 주었다.

표 4-10. DatagramSocket 클라스의 구성자

메 쏘 드	의 미
DatagramSocket( )	발송포구가 콤퓨터의 임의의 포구로 되는 UDP 소케트를 창조한다.
DatagramSocket(int port)	발송포구가 port인 UDP소케트를 창조한다.
DatagramSocket(int port, InetAddress address)	발송포구가 port이고 목적IP주소가 address인 콤퓨터에 UDP소케트를 창조한다.

DatagramSocket의 성원메쏘드를 표 4-11에 주었다.

표 4-11. DatagramSocket 클라스의 성원메쏘드

메 쏘 드	의 미
void close()	접속을 닫는다.
void connect(InetAddress address, int port)	발송포구가 port이고 목적콤퓨터의 IP가 address인 UDP접속을 진행한다.
void disconnect()	접속을 끊는다.
InetAddress getInetAddress()	소케트와 접속된 원격콤퓨터의 주소를 얻는다.
InetAddress getLocalAddress()	소케트와 련결된 호스트콤퓨터의 주소를 얻는다.
int getport()	원격콤퓨터의 포구번호를 얻는다.
int getLocalPort()	호스트콤퓨터의 포구번호를 얻는다.
void receive(DatagramPacket p)	데이터그램를 접수한다.
void send(DatagramPacket p)	데이터그램를 발송한다.
int getReceiveBuffersize()	수신완충기의 크기를 얻는다.
int getSendBufferSize()	발신완충기의 크기를 얻는다.
void setReceiveBufferSize(int size)	수신완충기의 크기를 설정한다.
void setSendBufferSize(int size)	발신완충기의 크기를 정한다.
int getSoTimeout()	제한시간을 얻는다.
void setSoTimeout(int timeout)	제한시간을 설정한다.

4.4.2. UDP규약을 리용한 실시간대화(Chat)프로그람의 작성

앞에서 설명한 UDP규약의 DatagramSocket방식에 기초한 통신프로그람을 작성해보자.

실례 4-5(UDP 규약을 리용한 봉사기프로그람의 작성)

1. Project Wizard를 사용하여 프로젝트를 창조

단계

1 【File】→【New Project】지령을 찰칵한다.

2 【Name:】본문칸에 《UDPServer》를 입력한다.

3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】→【New】지령을 찰칵한다.

2 Application아이콘을 두번 찰칵하면 Application Wizard대화칸이 펼쳐지는데【Title:】본문칸에 《Server Side》를 입력한다.

3 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.

2 Design태브를 찰칵한다.

3 부품선택판의 Swing서고를 선택한다.

4 여기서 jTextArea, jTextField, jButton부품 2개, jLabel를 선택하여 기본창문에 추가한다.

4. 부품의 속성값수정

단계

1 jButton1, jButton2부품의 text속성을 각각 《발송시작》, 《통보접수》로 고친다.

2 jTextField부품의 text속성에서 기정값을 지운다.

3 jLabel1부품의 text속성을 《통보:》로 고친다.

4 jTextArea부품의 text속성에서 기정값을 지운다.

5. 코드추가

Frame1.java의 import선언부분에

```
import.java.net.*;
import.java.io.*;
```

를 추가한다.

6. 프로그람의 작성

단계

1 jButton1부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton1_actionPerformed(ActionEvent e) {
  //통신에 쓰이는 UDP변수를 선언
  DatagramSocket socket = null;
  try
  {
    socket = new DatagramSocket(1801);
    byte[] buf = new byte[256];
      //요청접수
      DatagramPacket packet = new DatagramPacket(buf, buf.length);
      socket.receive(packet);
       //응답진행
      String dString =jTextField1.getText();
      dString.getBytes(0, dString.length(), buf, 0);
        //응답을 지정한 주소와 포구에 발송
      InetAddress address = packet.getAddress();
      //포구번호얻기
     int port = packet.getPort();
      //데이터그램창조
      packet = new DatagramPacket(buf, buf.length, address, port);
      //데이터그램발송
      socket.send(packet);
  }   //례외처리
  catch(IOException e1)
  {
```

```
    e1.printStackTrace();
  }
   //데이터그램닫기
  socket.close();
  jTextArea1.append("봉사기측 통보:"+jTextField1.getText()+" \n");
  jTextField1.setText("");
}
```

2 jButton2부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton2_actionPerformed(ActionEvent e) {
  try {
    //데이터그램창조,UDP소케트얻기
    DatagramSocket socket = new DatagramSocket();
    //요청발송
    byte[] buf = new byte【256】;
    InetAddress address = InetAddress.getByName("pio");
    DatagramPacket packet =
          new DatagramPacket(buf, buf.length, address, 1800);
    socket.send(packet);
    //봉사기응답얻기, 데이터그램접수
    packet = new DatagramPacket(buf, buf.length);
    socket.receive(packet);
    //접수자료현시
    String received = new String(packet.getData(), 0);
    jTextArea1.append("의뢰기측 통보:"+received);
    jTextArea1.append(" \n ");
    //DatagramSocket닫기
    socket.close();
  }
  catch (IOException ex) {
  }
}
```

7. 콤파일 및 실행

단계

1 【File】→【Save All】지령으로 프로젝트를 보관한다.

2 【F9】건을 눌러 콤파일 및 실행을 진행한다.

실행결과는 그림 4-13과 같다.

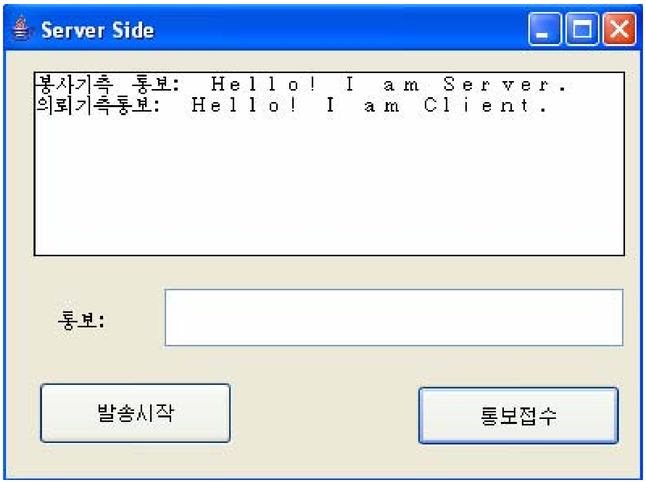


그림 4-13. UDP규약을 리용한 봉사기측 프로그람실례

실례 4-6(UDP 규약을 리용한 의뢰기측프로그람의 작성)

1. Project Wizard로 프로젝트를 창조

단계

1 【File】→【New Project】지령을 찰칵한다.

2 【Name:】본문칸값을 《UDPClient》로 고친다.

3 【Finish】단추를 찰칵한다.

2. Application Wizard리용

단계

1 【File】→【New】지령을 찰칵한다.

2 Application아이콘을 두번 찰칵하면 Application Wizard대화칸이 생기는데 【Title:】본문칸에 《Client Side》을 입력한다.

3 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.
2 Design태브를 찰칵한다.
3 부품선택판의 Swing서고를 선택한다.
4 jTextArea부품, jTextField부품, jButton부품 2개, jLabel부품을 추가한다.

4. 부품의 속성값수정

단계

1 jButton1과 jButton2부품의 text속성을 각각 《발송시작》, 《통보접수》로 설정준다.
2 jTextField1부품의 text속성에서 기정값을 지운다.
3 jLabel1부품의 text속성을 《통보:》로 수정한다.
4 jTexArea부품의 text속성에서 기정값을 지운다.

5. 변수설정

Frame1의 import구역에 아래의 내용을 추가한다.

```
import java.net.*;
import java.io.*;
```

6. 프로그람코드의 작성

단계

1 jButton1부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton1_actionPerformed(ActionEvent e) {
  //통신에 쓰이는 UDP변수를 선언
  DatagramSocket socket = null;
  try
  {
    socket = new DatagramSocket(1800);
    byte[] buf = new byte[256];
      // 요청접수
```

```
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
         //응답진행
        String dString = jTextField1.getText();
        dString.getBytes(0, dString.length(), buf, 0);
        // 응답을 지정한 주소와 포구에 발송
        InetAddress address = packet.getAddress();
        //포구번호얻기
        int port = packet.getPort();
        //데이터그램창조
        packet = new DatagramPacket(buf, buf.length, address, port);
        //데이터그램발송
        socket.send(packet);
    }   //례외처리
    catch (IOException e1)
    {
       e1.printStackTrace();
    }
    //데이터그램닫기
    socket.close();
    jTextField1.setText("");
}
```

2 jButton2부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton2_actionPerformed(ActionEvent e) {
    try {
          //데이터그램창조,UDP소케트얻기
      DatagramSocket socket = new DatagramSocket();
        //요청발송
        byte[] buf = new byte[256];
        InetAddress address = InetAddress.getByName("pio");
        DatagramPacket packet =
           new DatagramPacket(buf, buf.length, address, 1801);
        socket.send(packet);
        //봉사기응답얻기,데이터그램접수
```

```
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
        //접수자료현시
        String received = new String(packet.getData(), 0);
        jTextArea1.append("봉사기측통보:"+received);
        jTextArea1.append(" \n ");
        //DatagramSocket닫기
        socket.close();
    }
    catch (IOException ex) {
    }
}
```

7. 콤파일 및 실행

단계

1 【File】-【Save All】지령으로 프로젝트를 보관한다.

2 【F9】건을 눌러 콤파일 및 실행을 진행한다. 실행결과는 그림 4-14와 같다.

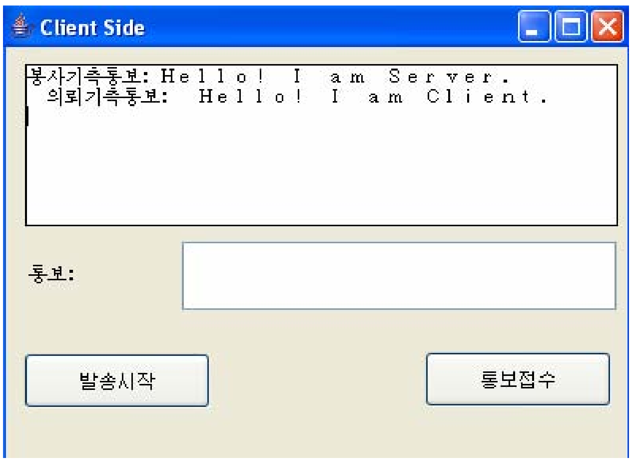


그림 4-14. UDP규약을 리용한 의뢰기측프로그람실례

제5절. FTP전송

Java에서는 망클라스서고 java.net.ftp를 리용하여 FTP접속을 실현한다. 이 클라스들을 리용하여 인터네트봉사기에 원격가입할수 있으며 파일열람, 전송 등도 진행할수 있다. 이 절에서는 FTP클라스서고의 FtpClient클라스를 통한 망프로그람작성방법에 대하여 서술한다.

4.5.1. FTP부품

FtpClient는 java.net.ftp클라스서고에서 가장 중요한 클라스로서 FTP의 거의 모든 기능을 포함하고있다.

- 먼저 이 클라스의 성원마당에 대하여 소개한다.

· public static boolean useFtpProxy

이 변수는 FTP전송이 대리봉사기를 리용하는가 하지 않는가를 나타낸다. 이 값이 True이면 대리봉사기를 사용하고있다는것을 나타낸다.

· public static String FtpProxyHost

이 변수는 useFtpProxy가 True일 때에만 유효한것으로서 대리봉사기의 이름을 표시한다.

· public static int FtpProxyPort

이 변수는 useFtpProxy가 true일 때에만 유효한것으로서 대리봉사기의 포구번호를 나타낸다.

-구성자를 리용하여 FTP접속을 창조한다.

· public FtpClient(String hostname, int port)

이 구성자는 지적된 봉사기이름과 포구를 리용하는 FTP접속을 창조한다.

· public FtpClient(String hostname)

이 구성자는 지적된 봉사기이름을 리용하여 FTP접속을 창조한다. 이때 포구는 기정포구를 리용한다.

· FtpClient()

이 구성자는 FtpClient객체를 창조할뿐 FTP접속을 진행하지 않는다. 이때 접속은 OpenServer메쏘드를 리용하여 진행한다.

- FtpClient클라스는 아래의 메쏘드를 리용하여 FTP봉사기와의 접속을 실현한다.

· public void openServer(String hostname)

이 메쏘드는 지정된 봉사기와 기정포구를 통하여 접속을 진행한다.

· public void openServer(string host, int port)

이 메쏘드는 지정된 봉사기와 지정된 포구번호를 리용하여 FTP접속을 진행한다. 접속

을 진행한 다음에는 FTP봉사기에 등록하여야 하는데 이때 아래의 메쏘드를 리용한다.

· public void login(String username, String password)

이 메쏘드는 파라메터 username과 password를 리용하여 FTP봉사기에 등록한다.

- 아래에서 FtpClient클라스가 제공하는 몇가지 메쏘드들을 소개한다.

· public void cd(String remoteDirectory)

remoteDirectory가 지적하는 등록부에로 이행한다.

· public void binany()

전송방식을 2진코드형식으로 설정한다.

· public void ascii()

전송방식을 ASCII코드형식으로 설정해준다.

· public TelnetInputStream list()

봉사기의 현재 등록부내용의 입력흐름을 얻는다.

· public TelnetInputStream get(String filename)

이름이 filename인 봉사기파일의 입력흐름을 얻는다. 이것을 리용하여 파일을 내리적재할수 있다.

· public TelnetOutputStream put(String filename)

filename에 대한 출력흐름을 얻는데 이 흐름을 리용하여 파일을 봉사기에 전송할수 있다.

4.5.2. FTP의뢰기측프로그람의 작성

아래에서 FTP프로그람작성실례를 고찰한다.

실례 4-7(FTP 의뢰기측프로그람)

1. Project Wizard를 리용하여 프로젝트를 창조

단계

1 【File】→【New Project】진행을 찰칵한다.
2 【Name:】본문칸에 《FTPClient》를 입력한다.
3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】 →【New】지령을 찰칵한다.
2 Application아이콘을 두번 찰칵한다.
3 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.

2 Design태브를 찰각한다.

3 부품선택판의 Swing서고를 선택하고 jTextArea부품, jTextField부품 3개, jCheckBox부품, List부품을 추가한다.

4. 부품의 속성값수정

단계

1 jButton1부품의 text속성을 《등록》으로 수정한다.

2 jTextField1, jTextField2, jTextField3부품의 text속성을 각각《192.168.8.51》, 《anonymous》, 《a》로 설정한다.

3 jCheckBox1부품의 text속성을 《별명》으로, State속성을 《true》로 준다.

4 List1부품의 text속성에서 기정값을 지운다.

5 jTextArea1부품의 text속성에서 지정값을 지운다.

5. 변수설정

Frame1의 import구역에 아래의 내용을 추가한다.

```
import java.io.* ;
import sun.net.TelnetInputStream;
import sun.net.ftp.* ;
```

그리고 변수선언부분에

```
FtpClient ftp=null;
```

를 추가한다.

6. 프로그람코드작성

단계

1 jButton1부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton1_actionPerformed(ActionEvent e) {
StringBuffer buf=new StringBuffer();
int ch;
```

```
list1.removeAll();
try {
  if (ftp!=null)
      ftp.closeServer();
}
catch (IOException ex) {
  ex.printStackTrace();
}
try {
  ftp= new FtpClient(jTextField1.getText());
  //Ftp등록
  ftp.login(jTextField2.getText(),jTextField3.getText());
  //ASCII규약사용
  ftp.ascii();
  //등록부표얻기
  TelnetInputStream t=ftp.list();
          t.setStickyCRLF(true);
          while((ch=t.read())>=0){
          if(ch=='\n')
            {
                 list1.add(buf.toString());
                 buf.setLength(0);
                 }else{
                 buf.append((char)ch);
            }
          }
          t.close();
        }
        catch (IOException ex) {
          ex.printStackTrace();
        }
        }
```

2 List1부품에 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void list1_mouseClicked(MouseEvent e) {
 StringBuffer buf=new StringBuffer();
 int ch;
 String dir=list1.getSelectedItem();
 //문자렬을 분해하여 등록부를 얻기
 int begin=0;
 int k=dir.length()-1;
 while(dir.charAt(k)==' ') k--;
 for(int i=k;i>1;i--){
   if(dir.charAt(i)==' ') {
     begin=i;
     break;
   }
 }
 jTextArea1.append("등록부: ");
 jTextArea1.append(dir.substring(begin));
 jTextArea1.append("\n");
try {
  ftp.cd(dir.substring(begin));
  TelnetInputStream t=ftp.list();
   while((ch=t.read())>=0)
   buf.append((char)ch);
   t.close();
      //등록부결과를 현시
      jTextArea1.append(buf.toString());
}
catch (IOException ex) {
  ex.printStackTrace();
}
}
```

3 jCheckBox1부품에 PropertyChange사건을 추가하고 아래의 코드를 입력한다.

```
void jCheckBox1_PropertyChange(PropertyChangeEvent e){
if(jCheckBox1. getState()){
jTextField1.setText( "anonymous" );
jTextField2.setText( "a" );
}
```

7. 콤파일 및 실행

단계

1 【File】→【Save All】을 리용하여 프로젝트를 보관한다.
2 【F9】건을 눌러 콤파일과 실행을 진행한다.

이 프로그람은 먼저 jTextField1에서 봉사기의 이름을 얻고 FTP접속을 진행한다.

```
ftp=new Ftp Cilent (jTextField1.getText());
// FTP에 등록
ftp.login(jTextField2.getText(), jTextField3.getText());
//ASCII규약을 리용
ftp.ascii();
```

다음 화면에 FTP의 등록부내용을 현시하며 사용자가 등록부에서 어느 한 등록부를 선택하면 jTextArea1에 그 등록부의 내용을 현시한다. 그림 4-15에 결과를 주었다.

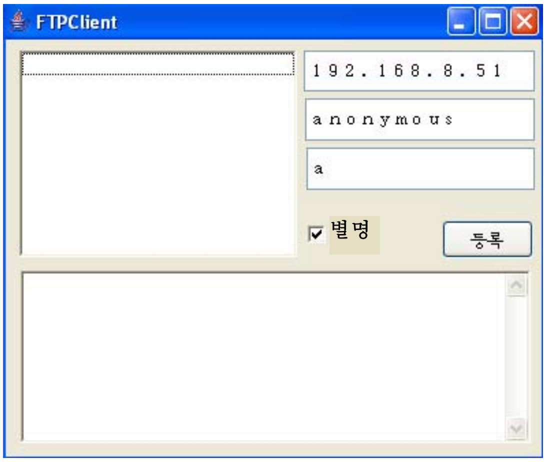


그림 4-15. FTP의 의뢰기측 프로그람실례

사실 FTP규약은 Java가 지원하는 TCP/IP표준규약의 한부분일뿐이다. getFile클라스로도 인터네트상의 FTP봉사기와 FTP접속을 실현하고 파일을 내리적재할수 있다. 이에 대해서는 서술하지 않는다.

제6절. Internet자원의 얻기

이 절에서는 인터네트의 자원을 리용하는 방법에 대하여 소개한다.

4.6.1. URL해석

URL은 문자렬형식으로 인터네트자원의 위치를 지정한다. 또한 URL은 자원의 위치와 자원에 접근하는데 리용되는 규약을 포함한다. URL은 Uniform Resource Locator의 략자로서 일반적으로 망에서의 파일이름이다.

URL은 아래의 부분들을 포함하고있다.

- 호스트콤퓨터이름: 자원이 있는 콤퓨터이름
- 규약이름: 자원에 접근하는데 리용되는 규약으로서 FTP, HTTP, Gopher, News 등이 있다.
- 파일보관위치: 호스트콤퓨터에서의 파일경로
- 포구번호: 자원과 접속하는데 리용되는 포구번호

Java망클라스서고의 URL클라스는 인터네트의 정보를 쉽게 얻게 하는 사용자응용프로그람작성대면부(API)이다. Java언어는 getHost, getPort, getProtocol, getFile메쏘드를 제공함으로써 URL을 분석할수 있게 한다. 아래의 실례에서는 4개의 API함수(getHost, getPort, getProtocol, getFile)들을 리용하여 지적된 URL의 상세한 내용을 분석하는 방법에 대하여 보여주었다.

실례 4-8(URL 클라스의 리용방법)

1. Project Wizard로 프로젝트를 창조

단계

1 【File】→【New Project】을 찰칵한다.
2 【Name:】본문칸에 《URL》를 입력한다.
3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】 →【New】지령을 찰칵한다.
2 Application아이콘을 두번 찰칵한다.
3 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Framel.java를 선택한다.
2 Design태브를 찰칵한다.
3 Swing의 jButton부품을 기본창문에 추가한다.
4 Swing의 jLabel부품을 4개 추가한다.
5 Swing의 jTextField부품을 추가한다.

4. jTextField1부품의 text속성에 《http://localhost:80/index.html》을 입력하고 jButton1의 text속성을 《URL정보조사》로 수정한다.

5. jButton1의 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton1_actionPerformed (ActionEvent e){
  try{
     URL myurl =new URL(jTextField1.getText());
     jLabel1.setText( "입력한 URL의 통신규약:" +myurl.getProtocol());
     jLabel2.setText( "입력한 URL의 호스트콤퓨터:" +myurl.getHost());
     jLabel3.setText( "입력한 URL의 파일이름:" +myurl.getFile());
     jLabel4.setText( "입력한 URL의 접속포구:" +myurl.getPort());
  }
  catch (MalformedURLException ex){
     ex.printStackTrace()}
  }
}
```

6. 콤파일 및 실행

단계

1 【File】→【Save All】지령으로 프로젝트를 보관한다.
2 【F9】건을 눌러 콤파일 및 실행을 진행한다.

실행결과는 그림 4-16과 같다.

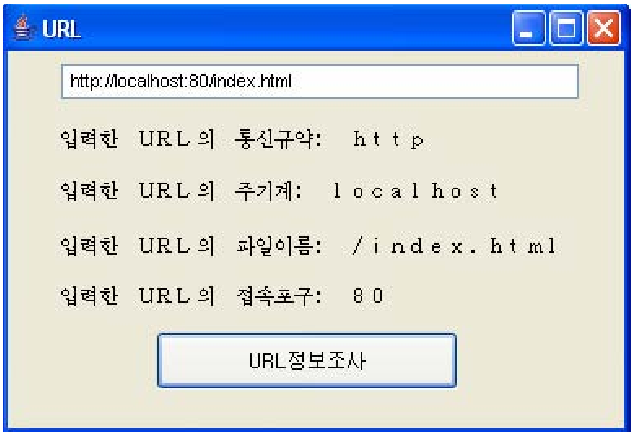


그림 4-16. URL분석실례

4.6.2. URLConnection클라스

URLConnection은 봉사기와 접속을 진행한 다음 파일을 내리적재하기 전에 그것의 속성을 알아보는데 리용된다. 이 속성들은 HTTP규약에 따라 규정되며 HTTP규약의 URL객체에 대해서만 의미가 있다. 실지 URLConnection클라스는 Java가 제공하는 규약처리클라스중의 하나이다. 이러한 클라스들에는 URLStreamHandler도 있다. 규약처리방법은 비교적 간단하다. URLConnection클라스는 추상클라스이므로 리용하려면 반드시 그의 하위클라스를 만들어야 한다. 일반적으로 아래와 같은 순서로 URLConnection클라스를 리용한다.

단계

1 URL객체를 창조한다.

2 URL객체의 openConnection()메쏘드를 리용하여 이 URL의 URLConnection객체를 검색한다.

3 URLConnection을 설치한다.

4 머리부마당을 읽는다.

5 입력흐름을 얻어 자료를 읽는다.

6 출력흐름을 얻어 자료를 읽는다.

7 접속을 닫는다.

URLConnection클라스의 유일한 구성자는 보호형식으로서 다음과 같이 정의한다.

protected URLConnection(URL url)

따라서 하위클라스를 창조하지 않는 경우에는 URL, URLStream클라스와 Handler, openConnection()메쏘드를 리용하여 URLConnection객체를 얻는다.

```
try{
    URL u=new URL("http://www.sec.com.kp");
    URLConnection uc=u.openConnection();
}
catch(MalformedURLException e){
    System.err.println(e);
}
catch(IOException e)
    System.err.println(e);
}
```

사실 java.net.URL의 openConnection()메쏘드와 java.net.URLStreamHandler의 openConnection()은 완전히 같다.

URLConnection클라스는 추상클라스이며 리용가능한 메쏘드는 connect()뿐이다.

그것의 봉사기와의 접속은 사용자가 실현하려는 봉사기의 형태에 따른다.(HTTP, FTP 등) 례를 들어 sun.net.www.protocol.file.FileURLConnection메쏘드는 URL을 해당한 등록부에 있는 파일이름으로 전환시키고 이 파일에 대한 MIME정보를 만들어준다. 다음 이 파일에 대한 FileInputStream완충기를 만든다. sun.net.www.http.Client는 connect()메쏘드를 리용하여 HttpClient객체를 창조한 다음 봉사기와의 접속을 진행한다. URLConnection을 창조했을 때에는 접속되지 않은 상태이다. 다시 말하여 자료를 전송하거나 받을수 없다. connect()메쏘드는 콤퓨터들사이의 접속을 진행하여 사용자가 자료통신을 진행할수 있게 한다. 이러한 접속은 보통 TCP소케트를 리용하여 실현한다. 그러나 getInputStream(), getContent(), getHeaderField()를 비롯한 일부 메쏘드들은 접속이 되여있을것을 요구한다. 만일 접속이 되여있지 않았으면 이 메쏘드들은 자동적으로 connect()메쏘드를 호출한다. 때문에 사용자가 직접 connect()메쏘드를 호출하는 경우가 드물다.

URLConnection클라스로 URL에서 자료를 검색하는 순서는 아래와 같다.

단계

1 URL객체를 창조한다.
2 URL객체의 openConnection()메쏘드로 URL의 URLConnection객체를 검색한다.
3 URLConnection의 getInputStream()메쏘드를 호출한다.

4 API의 흐름을 리용하여 입력흐름에서 자료를 읽어들인다.

getInputStream()메쏘드는 입력흐름을 귀환하므로 사용자는 봉사기가 보낸 자료를 읽고 분석할수 있다.

public InputStream getInputStream();

실례 4-9(URLConnection 메쏘드를 리용한 망페지의 내리적재)

설계순서는 아래와 같다.

1. Project Wizard로 프로젝트를 창조

단계

1 【File】→【New Project】지령을 찰칵한다.
2 【Name:】본문칸에 《URLConnection》을 입력한다.
3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】→ 【New】지령을 찰칵한다.
2 Application아이콘을 두번 찰칵한다.
3 【Finish】단추를 찰칵한다.

3. 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.
2 Swing의 jButton, jTextField, jScrollPane, jTextArea부품을 추가한다.
3 Swing의 jTextArea부품을 jScrollPane에 추가한다.

4. jTextField1의 text속성을 《http://localhost:80/index.html》로, jButton1의 text속성을 《망페지정보조사》로 고친다.

5. jButton1의 actionPerfonmed사건에 아래의 코드를 입력한다.

```
void jButton1_actionPerformed(ActionEvent e) {
  try {
      StringBuffer buf=new StringBuffer();
   //URLConnection클라스를 열어 읽기
```

```
      URL myurl=new URL(jTextField1.getText());
      URLConnection uc=myurl.openConnection();
      InputStream raw=uc.getInputStream();
      InputStream buffer=new BufferedInputStream(raw);
    //InputStream을 한개의 Reader와 련결
      Reader r=new InputStreamReader(buffer);
      int c;
     while((c=r.read())!=-1){
       buf.append((char)c);
     }
        jTextArea1.append(buf.toString());
   }
  catch (IOException ex) {
    ex.printStackTrace();
  }
}
```

6. 콤파일 및 실행

단계

1 【File】→【Save All】지령으로 프로젝트를 보관한다.

2 【Run】→【Run Project】지령 혹은 【F9】건으로 실행시킨다.

《망페지정보조사》단추를 찰칵하면 그림 4-17과 같은 결과가 얻어진다.

URL과 URLConnection클라스의 가장 큰 차이는 URLConnection에서 MIME머리부에 대한 접근과 HTTP1.0와의 호환성이다. URLConnection은 봉사기의 자료에 대하여 읽기쓰기를 진행할수 있다.

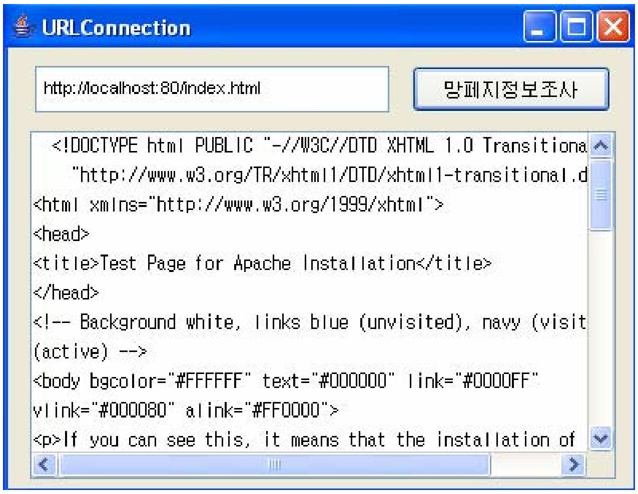


그림 4-17. 망페지의 보기

4.6.3. 전자우편의 통신방법

전자우편(E-mail)은 전자수단을 리용한 통신방법의 한 종류이다. 전자우편은 세계적으로 망에서 가장 많이 사용하는 봉사중의 하나로서 인터네트에서 HTTP규약 다음으로 사용률이 높다. JavaMail API는 J2EE의 한 부분으로서 소케트와 흐름을 리용한 순수한 Java API이다. JavaMail API를 리용하면 SMTP와 IMAP봉사기사이의 통신을 실현할수 있으며 전자우편을 발송하거나 접수할수 있다.

그림 4-18은 전자우편의 통신원리를 보여준다.

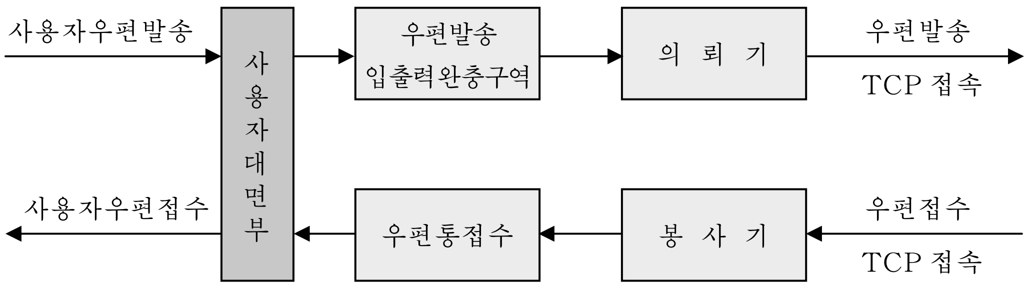


그림 4-18. TCP/IP 전자우편체계의 입출력완충원리

JavaMail API는 javax.mail패키지의 추상클라스들로 구성되여있다. 실례로 javax.mail.Message는 전자우편정보를 나타낸다. 이 클라스는 추상클라스로서 전자우편의 발신주소, 수신주소, 발신날자, 내용 등을 포함하고있다.

추상클라스인 javax.mail.Folder는 정보를 담는 용기의 역할을 한다. 이 클라스는 등록부정보얻기, 등록부들사이 이동정보, 지우기정보를 얻기 위한 기능들을 가지고있다. 이 추상클라스들은 전자우편의 보관, 콤퓨터들사이의 전자우편통신을 실현할수 없다. 표준인터네트전자우편을 리용하려면 javax.mail.Message대신에 javax.mail.MimeMessage를, javax.mail.Address대신에 javax.mail.InternetAddress를 리용하면 된다.

1) 전자우편의 발송

JavaMail API는 전자우편발송을 위한 모든 클라스와 메쏘드들을 제공한다.

전자우편을 발송하려면 아래의 단계를 걸쳐야 한다.

단계

1. mail.host속성을 설정하고 자기의 우편봉사기를 지정한다.
2. Session.getInstance()메쏘드를 리용하여 전자우편대화를 시작한다.
3. 새로운 Message객체를 창조하거나 그것의 하위클라스를 창조하여 실현할수 있다.
4. 우편의 발신주소를 설정한다.
5. 우편의 수신주소를 설정한다.
6. 전자우편의 주제를 쓴다.
7. 전자우편의 내용을 적는다.
8. Transport.send()메쏘드를 리용하여 전자우편을 발송한다.

첫 단계는 우편대화속성을 설정하는 단계로서 java.util.Properties객체를 만드는것이다. 아래의 코드는 mail.host를 mail.cloudy.net로 설정하는 코드이다.

```
Properties Props=Properties();
Props.put ( " mail.host " ,  "mail.cloudy.net" ) ;
```

사용자는 반드시 이 속성을 자기의 우편봉사기이름으로 설정하여야 한다. 이 속성은 Session.getInstance()를 사용하여 대화접속서고에서 Session객체를 검색하는데 리용한다. 상응한 코드는 아래와 같다.

```
Session mailConnection = Session.getInstance(Props, null);
```

Session객체는 프로그람과 전자우편봉사기사이에서 진행되는 통신을 나타낸다.

getInstance()메쏘드의 두번째 파라메터(여기에서는 null)는 javax.mail.Authenticator로서 암호가 필요할 때 사용자에게 발송된다.

Session객체는 새로운 Message를 만들 때 리용한다. 상응한 코드는 아래와 같다.

```
Message  msg = new MimeMessage(mailConnection);
```

사용자는 발신주소와 수신주소를 설정해주어야 하는데 이것은 모두 java.mail.internet.InternetAddress객체를 리용한다. 또한 사용자는 전자우편주소 혹은

이름을 줄수 있다. 례를 들어

Address kyc=new InternetAddress(pio@sec.com, "kim yong chol");

Address kuh=new InternetAddress(pio@sec.com);

setFrom()메쏘드는 사용자가 발신주소의 머리부를 설정해주므로 수신자에게 혼돈을 줄수 있게 한다. 다시말하여 Msg.setFrom(kyc);을 리용하여 《kim yong chol》로 된 수신주소를 위조할수 있다.

setRecipient()메쏘드에는 발송목적지의 주소와 목적지에 가닿는 경로를 주어야 한다. 이것은 아래에 준 Message.RecipientType클라스의 3개의 상수를 리용하여 표시한다.

Message.RecipientType.TO

Message.RecipientType.CC.

Message.RecipientType.BCC

주제는 다음과 같이 준다.

Msg.setSubject ("어머니앞");

내용은 문자렬로서 MIME형이여야 한다. 자주 쓰는 형태는 text/plain이다. 실례로

msg.setContent("래일저녁 평양도착", "text/plain");

마지막으로 Transport.send()메쏘드는 mail.host의 속성으로 지정된 우편봉사기와 접속하여 전자우편을 발송한다. 그 방법은 아래와 같다.

Transport.send(msg);

아래에 전자우편을 발송하는 간단한 실례를 주었다.

실례 4-10(전자우편의 발송)

1. Project Wizard로 프로젝트를 창조

단계

1 【File】→【New Project】지령을 찰칵한다.

2 【Name:】본문칸에 《SendMail》을 입력한다.

3 【Finish】단추를 찰칵한다.

2. Application Wizard의 리용

단계

1 【File】→ 【New】 지령을 찰칵한다.

2 Application아이콘을 두번 찰칵한다.

3 【Finish】단추를 찰칵한다.

3. 프레임설정과 조종부품의 추가

단계

1 프로젝트판의 Navigation에서 Frame1.java를 선택한다.

2 Design태브를 찰칵한다.

3 Swing의 jLabel부품 3개, jButton부품, jTextField부품 3개, jScrollPane부품을 프레임에 추가한다.

4 Swing의 jTextArea부품을 jScrollPane1에 추가한다.

4. jTextField1의 text속성을 《“kc@sec.com》으로, jButton1의 text속성을 《발송》으로 한다.

5. jButton의 actionPerformed사건을 추가하고 아래의 코드를 입력한다.

```
void jButton1_actionPerformed(ActionEvent e) {
  try{
     Properties props = new Properties();
     props.put("mail.host","pio.sec.com");
     Session mailConnection = Session.getInstance(props,null);
     Message msg = new MimeMessage(mailConnection);
     Address sendman=new InternetAddress("kc@sec.com");
     Address recieveman = new InternetAddress(jTextField1.getText());
     msg.setContent(jTextArea1.getText(),"text/plain");
     msg.setFrom(sendman);
     msg.setRecipient(Message.RecipientType.TO,recieveman);
     msg.setSubject(jTextField4.getText());
     Transport.send(msg);
     jTextArea1.setText("우편발송완성!");
   }
   catch (Exception ex){
     ex.printStackTrace();
   }
}
```

6. 콤파일 및 실행

단계

1 【File】→【Save All】지령으로 프로젝트를 보관한다.

2 【Run】→【Run Project】지령 혹은 【F9】건을 리용하여 콤파일 및 실행을 진행한다. 《발송》단추를 찰칵하면 그림 4-19와 같은 결과가 나타난다.

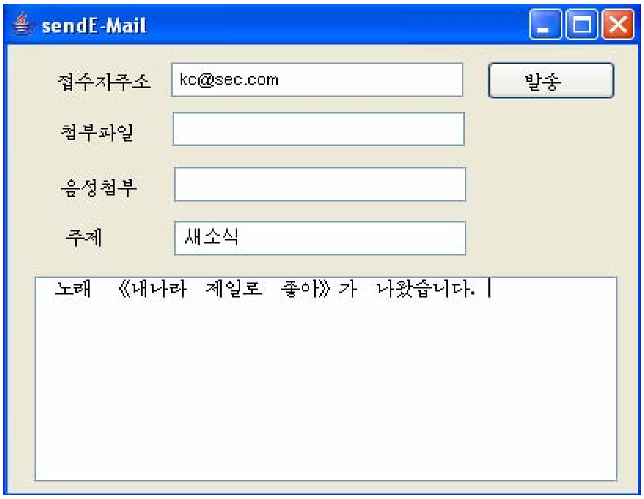


그림 4-19. 전자우편의 발송실례

2) 전자우편의 접수

전자우편의 접수는 많은 단계를 거쳐야 한다.

단계

1 접속속성을 설정한다.
2 접속에 리용할 Authenticator를 만든다.
3 Session.getDefaultInstance()를 리용하여 Session객체를 얻는다.
4 Session의 getStore()메쏘드를 리용하여 Store를 얻는다.
5 Store와 접속한다.
6 getFolder()메쏘드를 리용하여 Store로부터 INBOX서류철을 얻는다.
7 INBOX서류철을 연다.
8 접근하려는 INBOX서류철의 하위서류철을 연다.
9 서류철에서 Message객체를 배렬형식으로 얻는다.
10 Message클라스의 메쏘드를 리용하여 필요한 조작을 진행한다.
11 서류철을 닫는다.
12 보관구역을 닫는다.

아래에 간단한 POP3의뢰기측응용프로그람을 주었다.

```
import javax.mail.*;
import javax.mail.internet.*;
import java.util.*;
......................................
  void jbutton1_actionPerformed(ActionEvent e) {
    Properties props = new Properties();
    String host = textField1.getText();
    String username = textField2.getText();
    String password = textField3.getText();
    String provider = textField4.getText();
    try{
       //POP3봉사기와 련결
       Session session = Session.getDefaultInstance(props,null);
       Store store= session.getStore(provider);
       store.connect(host,username,password);
       //서류철열기
       inbox = store.getFolder("INBOX");
       if(inbox == null){
         System.out.println("No INBOX");
         System.exit(1);
       }
       inbox.open(Folder.READ_ONLY);
       list1.removeAll();
       //봉사기로부터 통보문얻기
       Message[] messages = inbox.getMessage();
       for(int i=0;i<messages.length;i++){
         list1.add("우편"+String.valueOf(i+1)+":"+messages[i].getSubject());
      }
     //닫기, 봉사기의 통보문을 삭제하지 않음
     //inbox.close(false);
     store.close();
   }
    catch(Exception ex){
                 ex.printStackTrace();
   }
  }
}
```

제5장. Applet개발기술

Java프로그람은 일반적으로 2가지 류형 즉 Application와 Applet가 있다. Applet는 인터네트에서 2진코드프로그람을 내리적재하여 그것을 자기의 콤퓨터에서 실행시킬수 있는 작은 응용프로그람을 말한다. Applet는 망환경에서 사용자가 실행시키는 프로그람이므로 의뢰기측프로그람이라고도 말한다.

이 장에서는 JBuilder에서 Applet프로그람의 작성방법에 대하여 학습한다.

제1절. Applet의 작업원리

Applet는 일종의 Java프로그람으로서 Java를 지원하는 웨브열람기에서 실행할수 있다. Applet를 실행하는데 필요한 대부분의 도형지원(례하면 단추, 표식자, 본문입력칸 등)능력을 열람기자체가 가지고있다. Applet프로그람은 웨브페지의 Applet HTML태그에 의하여 실행된다. Java를 지원하는 웨브열람기나 JDK의 AppletViewer를 리용하면 Applet의 실행결과를 볼수 있다.

다음의 실례를 고찰하자. 이것은 간단한 HTML파일이다.

```
<html>
<head>
<meta http-equiv= “Content-Type” content= “text/html;charset=Big5” >
</head>
<body>
firstApplet.Applet1 will appear below in a Java enabled browser.<br>
<Applet
  codebase= “.”
  code  = “firstApplet.Applet1.class”
  name  =  “TextApplet”
  WIDTH  = “400”
  height  = “300”
  hspace  = “0”
  align  = “middle”
>
```

```
</Applet>
</body>
</html>
```

열람기에서 이 파일을 보면 이름이 《firstApplet.Applet1.Class》인 Applet응용프로그람이 실행된다. Applet프로그람은 열람기에 의해 조종되므로 main()메쏘드가 필요없다.

먼저 Applet견본프로그람을 보자. 프로그람의 원천코드는 아래와 같다.

```
package fitstApplet;
import java.awt.*;
import java.awt.event.*;
import java.Applet.*;
public class Applet1 extends Applet{
  private Boolean isStandalone=false;
  public String getParameter(String key,String def){
    return isStandalone ? System.getProperty(key,def):
     (getParameter(key)!=null ? getParameter(key):def);
  }
  public Applet1(){
  }
  public void init(){
    try{
       jbInit();
    }
    catch(Exception e){
      e.printStackTrace();
    }
   }
   private void jbInit() throws Exception{
   }
   public String getAppletInfo(){
     return null;
   }
}
```

보는바와 같이 프로그람의 앞부분에 java.Applet.Applet를 적재하였다. JDK에서 java.Applet.Applet는 java.awt.Panel을 계승한것이다.

Panel은 AWT의 용기이므로 배치관리기를 가지고있는데 기정관리기는 Flowlayout 배치관리기이다. java.awt.Panel의 클라스는 java.awt.Container에서, java.awt.Container는 java.awt.Componet에서 계승된것이므로 Applet는 하나의 부품이라고 할수 있다. 그러므로 Applet에서는 각종 사건처리를 할수 있으며 다른 용기를 추가할수 있다.

일반적으로 Applet프로그람은 아래의 4개 메쏘드에 의하여 조종된다.

- init()—Applet가 들어있는 홈페지를 펼칠 때 init()메쏘드를 리용하여 Applet를 초기화한다.
- start()—Applet가 들어있는 홈페지를 펼칠 때 init()메쏘드를 호출한 다음 Start()메쏘드를 리용하여 Applet를 기동한다.
- stop()—Applet를 닫을 때 stop()메쏘드를 리용한다. stop()은 항상 destroy()메쏘드앞에서 호출된다.
- destroy()—stop()을 호출한 다음 destroy()를 사용하여 리용하고있던 자원을 정리한다.

그림 5-1은 간단한 Applet의 실행과정을 보여준다.

열람기는 HTML파일을 불러들여 〈Applet...〉를 포함하고있는가를 찾아본 다음 있으면 Applet 2진코드(bytecode)를 내리적재한다. 다음 JVM(Java Virtual Machine;Java 가상기계)에 넘겨준다. Java가상기계는 init()메쏘드를 찾는다. 만일 찾지 못하면 자체의 init()를 호출하고 paint()를 자동적으로 실행한다.

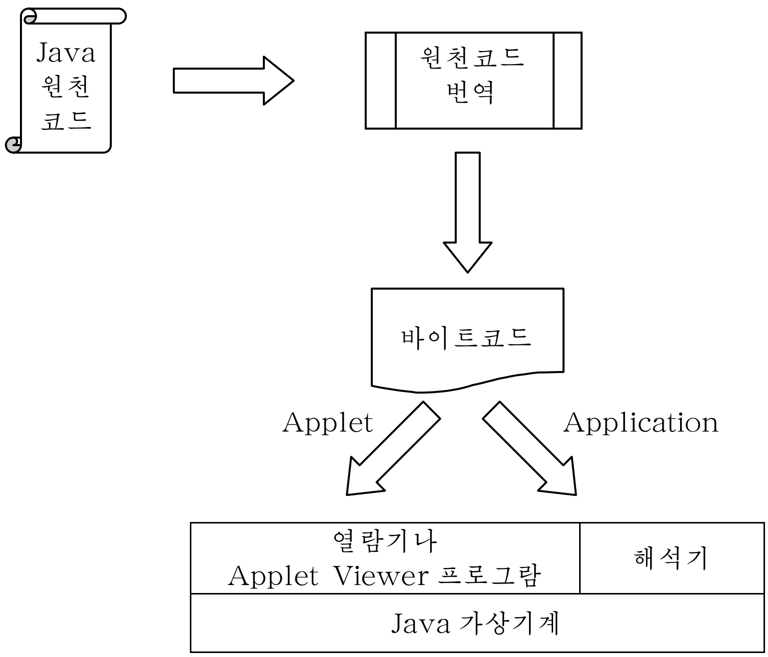


그림 5-1. Applet작업원리

제2절. Applet태그

아래에서는 APPLET태그의 형식을 보여주고있다. 반드시 있어야 할 요소는 굵은체로 표시하고 선택가능한 요소는 보통체로 표시하였다.

```
<APPLET
        CODEBASE = codebaseURL
        ARCHIVE = archiveList
        CODE = AppletFile ... or ... OBJECT = serializedApplet
        ALT = alternateText
        NAME = AppletInstanceName
        WIDTH = pixels HEIGHT = pixels
        ALIGN = alignment
        VSPACE = pixels HSPACE = pixels
>
<PARAM NAME = AppletAttribute1 VALUE = value>
<PARAM NAME = AppletAttribute2 VALUE = value>
...
alternateHTML
</APPLET>
```

CODE, CODEBASE 등은 Applet의 속성으로서 열람기에서 Applet에 대한 정보를 제공한다. 반드시 필요한 속성은 CODE, WIDTH, HEIGHT들이다.

아래에서 매 속성들에 대하여 소개한다.

- CODEBASE=codebaseURL

이 속성은 Applet의 기본URL 즉 Applet코드목록을 지정해준다. 만일 이 속성을 지정해주지 않으면 해당 HTML문서의 URL을 리용한다.

- ARCHIVE=archiveList

이 속성은 미리 적재하여야 할 클라스 혹은 기타 자원들을 포함하는 보존파일들을 지정한다. AppletClassLoader를 리용하는 구체례는 CODEBASE를 리용하여 이 클라스들을 적재한다. archiveList의 보존파일들이 여러개일 때에는 반점(,)으로 구분한다. 그리고 archiveList의 보존파일들은 CODEBASE와 같은 등록부 혹은 그의 부분등록부에 있다.

- CODE=AppletFile

이 속성은 반드시 필요한 속성으로서 Applet와 그의 하위클라스들이 콤파일된 다음에 생성되는 바이트코드파일이름이다. 이 파일은 Applet에 대응하는 기본URL로서 절대적인 URL로 지정되지 않는다. CODE와 OBJECT중 하나는 반드시 있어야 한다. AppletFile의 형식은 classname.class 혹은 packagename.classname.class로 될수 있다.

- OBJECT=SerializedApplet

이 속성은 Applet를 직렬화한 파일이름이다. init()메쏘드는 호출되지 않을수 있지만 start()메쏘드는 반드시 호출되게 된다.

Applet구체례에 넘겨주는 임의의 속성들은 모두 Applet에서 리용할수 있다. 그러나 직렬화하기전에는 반드시 Applet를 정지하여야 한다. 코드는 다음과 같다.

- ALT=alternateText

이 속성은 열람기가 Applet의 태그부호를 식별할수 있지만 Java Applet를 실행할 때 내용을 현시하지 못한다는것을 지적한다.

- NAME=AppletInstanceName

이것은 Applet의 이름을 지적하여 같은 망페지의 Applet가 호상 접근할수 있게 한다.(병렬통신)

- WIDTH=pixels HEIGHT=pixels

이것은 반드시 필요한 속성으로서 Applet가 현시되는 구역의 너비와 높이를 규정한다.

- ALIGN=alignment

이것은 Applet의 배치방식을 지정한다. 이 속성에는 left, right, top, texttop, middle, absmiddle, baseline, bottom, absbottom이 있다.

- VSPACE=pixels HSPACE=pixels

이 속성들은 Applet의 우,아래(VSPACE)와 량쪽(HSPACE)의 너비를 말한다.

- param태그

〈param name=AppletAttribute1 VALUE=Value〉
〈param name=AppletAttribute2 VALUE=Value〉

이 태그는 Applet의 특정한 속성을 지정할수 있는 태그이다. Applet는 getParameter() 메쏘드를 리용하여 이 태그에서 지정한 특정한 속성들을 얻는다.

Java를 지원하는 웨브열람기에서는 Applet프로그람을 실행시킬 때 Applet실행에 반드시 필요한 내용들을 제공해준다. 례를 들어 Applet를 포함하고있는 웨브페지를 열람할 때 열람기는 웨브페지를 초기화하고 Applet프로그람을 기동한다. 또한 이 웨브페지를 끄면 열람기는 상응한 메쏘드를 호출하여 Applet프로그람의 실행을 끝마친다.

제3절. Applet와 열람기

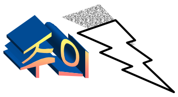
일반적으로 Applet는 AppletViewer 또는 Java를 사용하는 열람기에서 실행할수 있다. 물론 새로운 API는 낡은 판본의 API열람기에서 실행할수 없다. 때문에 판본제한을 받게 된다.

AppletViewer지령의 사용방식은 다음과 같다.

AppletViewer [options] urls...

이 지령은 AppletViewer가 url로 지정된 원천과 련결하여 Applet를 현시하게 한다.

추가선택항목 -debug는 Java오유제거기에서 AppletViewer가 기동할 때 Applet의 오유제거를 할수 있게 한다. 추가선택항목 -encoding(부호화)은 HTML파일의 부호화이름을 가리킨다. 추가선택항목 -J javaoption은 javaoption문자렬이 파라메터로서 AppletViewer의 Java해석기에 전달된다. 파라메터는 공백을 포함하지 말아야 한다. 여러개의 파라메터들로 이루어진 문자렬은 매 파라메터앞에 -J를 붙여야 한다. 이것은 오유제거와 기억기사용에서 매우 유용하다.



만일 url이 지정하는 파일이 OBJECT, EMBED 또는 APPLET태그를 포함하지 않으면 AppletViewer는 아무런 효과도 나타내지 못한다. AppletViewer에서는 Applet기동에 관계없는 HTML태그들은 무시한다. AppletViewer가 식별할수 있는 HTML태그를 표 5-1에 제시하였다.

표 5-1. AppletViewer가 지원하는 태그

태그	적용과 사용법
object	Object태그는 Applet를 HTML페지에 삽입하는 HTML4.0의 태그이다. <object WIDTH= “pixelWIDTH” height= “pixelHeight” > <param name= “code” value= “youClass.class” > <param name= “object” value= “serializedObjectOrJavaBean” > ... alternate-text </object>

embed

embed태그는 Netscape의 HTML3.2에 대한 확장기능으로서 HTML페지에 Applet나 다매체대상을 삽입가능하게 한다.

```
<embed
  code= "yourClass.class"
  object= "serializedObjectOrJavaBean"
  codebase= "classFileDirectory"
  WIDTH= "pixelWIDTH"
  height= "pixelHeight"
>
  ...
</embed>
```

Applet

Applet태그는 HTML3.2의 태그로서 HTML페지에 Applet를 삽입하는데 리용한다. 이것을 리용하여 내리적재된 Applet를 열람기에서 실행한다. 최신판의 Java가동기반에서는 object태그를 리용하여 Java plugin을 내리적재하여야 한다.

```
<Applet
    code= "youClass.class"
    object= "serializedObjectOrJavaBean"
    codebase= "classFileDirectory"
    width= "pixelWIDTH"
    height= "pixelHeight"
>
  <param name= "..." value= "..." >
  ...
  alternate-text
</Applet>
```

app

app태그는 Applet의 간략형이다. 현재 app를 쓰지 않고 Applet태그를 리용하고있다.

```
<app
  class= "classFileName" (without a.class suffix)
  src= "classFileDirectory"
  WIDTH= "pixelWIDTH"
  height= "pixelHeight"
>
 <param name= "..." value= "..."
 ...
</app>
```

5.3.1. Java의 지원

앞에서 본바와 같이 Applet프로그람과 열람기사이의 JDK의 판본이 맞지 않을수 있다. 레를 들어 많은 열람기들이 JDK1.1.7 Swing을 지원하지 못한다. 이때 클라스적재기(ClassLoader)는 열람기의 JDK가 Java클라스 혹은 메쏘드를 지원할수 없기때문에 《No Class Def Found Error》와 같은 오유들을 발생시킨다. 또한 JDK1.2와 JDK1.3들도 모든 열람기가 다 지원하는것은 아니다. 때문에 Applet작성자는 Applet프로그람이 지원하는 JDK판본을 알아야 한다.

열람기가 지원하는 JDK판본은 열람기의 Java조작탁(Console) 또는 열람기의 웨브페지에서 찾아볼수 있다. 례를 들어 Netscape의 【Communicator】차림표의 【Tools】→【Java Console】지령, Internet Explorer의 【Tools】→【Sun Java Console】지령을 통하여 JDK판본을 알수 있다. 만일 Sun회사의 Java Plug-in을 사용한다면 같은 판본의 JDK를 사용자말단들에 내리적재하는 방법으로 JDK판본문제를 간단히 해결할수 있다.

5.3.2. 말단사용자의 열람기선택

먼저 말단사용자는 적합한 판본(Applet가 정상적으로 실행될수 있는 가장 낮은 판본)의 열람기를 설치하여야 한다. 이미 열람기가 설치되여있다면 반드시 Applet를 사용할수 있도록 적합한 판본의 열람기로 갱신하여야 한다. 이때 열람기의 갱신판본 혹은 보충설치프로그람을 내리적재하여 설치하는 방법으로 JDK판본문제를 해결할수 있다.

5.3.3. 여러 열람기로부터의 지원

만일 Applet를 여러 열람기에서 실행할수 있도록 하려면 반드시 열람기들사이의 차이를 알아야 한다. 될수록 모든 열람기들에서 실행해보고 그 결과를 분석해보아야 한다.

5.3.4. Java실현에서의 차이

일부 열람기들은 사용지도서와 규범들을 필수적으로 제공하고있다. 열람기제작자들은 이러한 지도서와 규범들을 지키면서 제작하려는 열람기에서 그 기능을 확장한다. 따라서 서로 다른 열람기에서 Java를 실행하면 차이가 생기며 Applet를 각이한 환경에서 어떻게 정상적으로 실행하겠는가 하는것이 하나의 문제로 제기된다. 실례로 보안관리와 보안급수별 Java실현에서 차이가 생긴다. 중간급의 보안급수를 가진 열람기에서 실행하는 프로그람은 다른 열람기에서는 집행되지 않는다.

이와 같이 의뢰기에 따라 보안급수를 조절하여야 한다. 이때 서명기구(signature mechanism)를 리용하면 Applet를 실행하는데서 보다 유연한 실행환경을 구성할수 있다. keytool과 jarsigner는 이 기구를 사용하지만 열람기들 모두가 이 기구를 지원하는것은 아니다.

일부 열람기들에서는 자기의 보안기구를 자기의 열람기에서만 사용할수 있게 하였다. 실례로 Sun, Netscape Communicator와 IE들은 각각 서로 자기의 Applet서명기구를 제공한다. 그러나 그 원리들은 모두 서로 비슷하다. 즉 믿을만한 서명자의 수자서명에 대한 검증을 통하여 Applet가 믿을수 있는 사람이 제공한것인가를 확인한다.

Sun회사에서는 JDK1.X판본에 javakey라고 하는 작은 프로그람을 제공하고있다. 이것은 개발자를 대신해서 Applet에서 리용하려는 수자서명의 모든 단계를 완성할수 있게 하며 또한 최종사용자를 대신해서 개발자가 신임하는 작업을 완성할수 있다.

열람기들에 따라 Java실행환경들이 각이하다. 실례로 마이크로쏘프트회사에서는 필요에 따라 Java가상기계를 수정한다. IE는 Java의 일부 클라스서고를 포함하고있지 않으므로 Applet가 IE에서 정상적으로 실행될수 없다. 결국 같은 Applet가 서로 다른 열람기에서 실행될 때의 결과가 다를수 있다. 또한 같은 열람기라도 조작체계에 따라 Java에 대하여 지원하는 정도가 다르다. 실례로 서로 다른 조작체계에서 스레드에 대한 지원은 서로 다르며 따라서 여러개 스레드의 Applet실행결과도 같지 않다. 례를 들어 마이크로쏘프트회사에서 새로 내놓은 조작체계 Windows XP는 Java를 지원하지 않는다.

바로 이러한 원인으로 하여 Applet를 시험할 때에는 의뢰기의 실제적인 실행환경을 고려하여 여러가지 환경에서 시험을 진행하여야 한다.

5.3.5. 열람기판본문제에 대한 해결

아래와 같은 몇가지 방법으로 열람기의 판본문제를 해결할수 있다.

- Java끼워넣기 리용

Sun회사는 열람기문제를 해결하기 위하여 Java끼워넣기를 도입하고 이미 Java Plug-in이라고 부르는 프로그람을 개발하여 Applet를 Navigator나 IE에서 사용할수 있는 Java가동기반을 만들었다. 그러나 여기서 중요한 문제는 Java끼워넣기를 실현하려면 사용자콤퓨터에 가상기계를 설치하여야 한다는것이다.

Java Plug-in을 리용하면 홈페지내용의 현시와 Java Applet의 실행작업을 분리하여 진행할수 있다. 열람기는 다만 대면부프로그람으로서 홈페지의 내용을 읽어들이고 그의 내용(영상, 문자, Applet 등)들을 현시하는 기능을 수행한다. Applet실행작업은 다른 프로그람에서 하므로 열람기에서 처리하여서는 안된다.

- 열람기의 사용을 지원하는 같은 종류의 JDK

비교적 믿음성있는 방법은 열람기의 사용을 지원하는 같은 판본의 JDK를 사용하는것이다. 그러면 JDK환경의 차이를 막을수 있다.

- JDK 1.1.X 또는 이전판의 JDK사용

JDK 1.1.X 또는 이전판의 JDK를 리용하여 Applet를 편집하면 각이한 판본의 열람기에서 생길수 있는 문제를 해결할수 있다. 그러나 이런 방법을 쓰면 Applet는 최신판본의 JDK가 제공하는 풍부한 클라스서고와 기본도구 및 최신기능을 리용할수 없다.

- 같은 종류의 열람기와 서로 같은 조작체계의 사용

실제로 의뢰기말단에서 같은 종류의 열람기를 리용하면 열람기가 지원하는 JDK와 조작체계가 서로 같다. 의뢰기말단에서 이러한 환경이 보장되면 Applet는 정상적으로 실행될수 있다.

- Java Web Start의 사용

Java Web Start(JWS)는 Sun회사에서 웨브를 통하여 Java프로그람을 배포하는 새로운 기술로서 이것을 리용하여 Application와 Applet를 배포할수 있다. 이것은 첫 실행시 프로그람을 내리적재하여 JWS에 넘겨주어 판본의 자동갱신과 관리를 진행하게 한다. 프로그람은 사용자말단에서 실행되지만 사용자말단에 설치할 필요가 없다. 또한 판본이 갱신된다음에는 의뢰기측에서 수정할 필요가 없다. 이것이 바로 JWS의 우점의 하나이다.

JWS는 주로 망상에서 배치되고 응용되는 프로그람으로서 보안성이 높으며 관리비용이 적고 사용하기 쉬운 특징을 가지고있다.

사용자는 JWS를 리용한 응용프로그람망봉사기에서 프로그람을 내리적재할수도 있고 그대로 실행할수도 있다. 또한 JWS의 의뢰기말단을 통하여 이미 내리적재된 응용프로그람을 원격으로 실행할수도 있다. 같은 응용프로그람에 대해서는 첫번째 실행시 내리적재하고 그 다음 매번 실행할 때는 JWS의 의뢰기말단이 자동적으로 판본을 알아보고 판본이 갱신되였으면 자동적으로 의뢰기의 판본을 갱신한다. 다시말하여 사용하기 불편한 부분들을 JWS가 맡아 처리한다.

5.3.6. Applet에서 알아야 할 추가적인 문제들

1) 모래통(sand box)기구에 의한 Applet의 보안

열람기 또는 AppletViewer에서 원격콤퓨터에 있는 Applet를 실행할 때 Java가상기계는 그것을 믿을수 없는것으로 보고 그것을 모래통의 보호속에 넣는다. 이러한 경우에 Applet의 실행은 다음과 같은 제한을 받게 된다.

- Applet는 의뢰기콤퓨터의 파일을 읽기쓰기할수 없다.
- Applet에서는 JCE/JCA의 addProvider조작을 진행할수 없다.
- 망에 련결할 때 Applet는 그것을 제공하는 봉사기와만 련결을 실현할수 있다.
- Applet는 사건의 포착, 체계정보의 수집 등 기타 방면에서 제한을 받는다.
- Netscape열람기에서는 java.security.*의 지원을 제공하지 않으며 IE열람기에서는 Applet의 실행에서 ApplerViewer에 비하여 더 높은 제한을 요구한다.

우에서 설명한것과 같은 제한으로 하여 보안이 된 공문전달체계를 열람기 또는 ApplerViewer에서 실행할 때 다음과 같은 문제들이 제기될수 있다.

- 기억기에서 읽기쓰기를 할수 없다.
- addprovider조작을 집행할수 없다.
- 추가적인 서고를 사용할 때 Security Exception례외가 발생한다.

· Netscape열람기에서는 java.security.*의 지원이 없으므로 실행할수 없다.

IE열람기에서는 Security Exception례외가 많이 발생하게 되는데 이 원인은 각종 Java 가상기계들이 공문전달체계의 의뢰기프로그람을 믿을수 없는것으로 보기때문이다. 해결방법은 Java가상기계가 공문전달체계의 의뢰기프로그람을 믿게 하는것이다.

2) Applet를 개발하는 과정에 주의하여야 할 문제들

정확한 프로그람을 작성, 편집하는 습관을 키워야 한다.

적합한 열람기 및 그에 따르는 JDK를 선택하여야 한다.

Java의 대소문자쓰기 규정에 주의하여야 한다.

개발된 파일을 정확한 경로에 배치하여야 한다.

정기보존파일을 효과적으로 사용하여야 한다.

될수록 패키지(package)파일을 리용하여야 한다.

제4절. JBuilder에 의한 Applet개발

JBuilber는 다음과 같은 Applet개발을 위한 몇가지 도구들을 제공한다.

- Applet조수
- 사용자대면부설계도구에 의한 Applet개발
- JBuilber를 리용한 Applet의 시험
- Sun회사가 제공하는 AppletViewer도구

5.4.1. Applet조수의 사용

JBuilber는 Applet조수를 제공하고있다. Applet조수를 리용하면 Java Applet프로그람을 빨리 개발하고 실행할수 있으며 동시에 기초코드를 자동적으로 생성할수 있다. 개발자는 일부 필요한 코드만을 추가하여 프로그람을 작성,완성할수 있다.

아래에서는 조수의 리용에 대하여 구체적으로 설명한다.

단계

1 【File】→【New Project】지령을 선택하여 Project Wizard대화칸을 연다. 사용자는 필요에 따라 Project Wizard대화칸의 첫 페지에서 프로젝트와 등록부 이름을 수정할수 있다. 【Generate project notes files】검사칸을 선택한다.



그림 5-2. Project Wizard대화칸의 첫 페지

2【Next】단추를 찰칵하여 Project Wizard대화칸의 다음 페지에 들어간다.(그림 5-3) 이 페지에서 일부 추가선택항목은 보통 기정으로 선택한다. 【JDK】에서 각이한 JDK판본을 선택입력할수 있고 기타 마당【Backup Path】,【Output Path】,【Working directory】에도 해당한 내용들을 입력할수 있다.

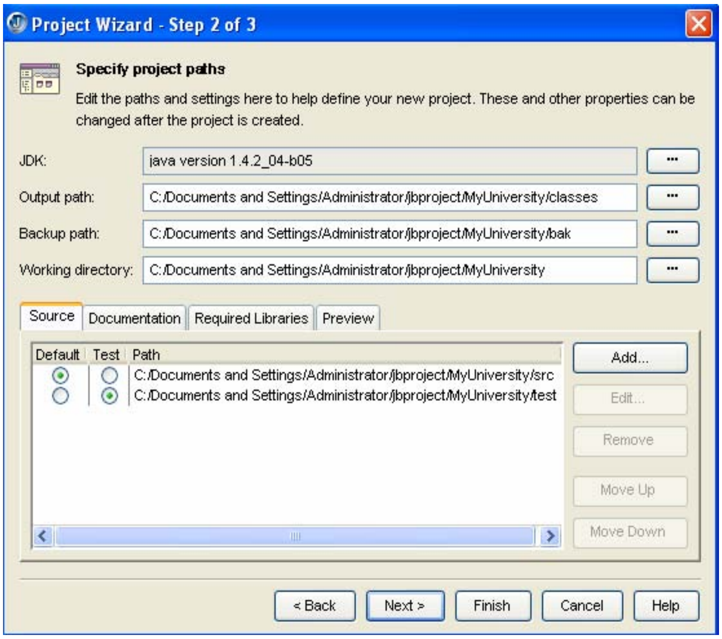


그림 5-3. Project Wizard대화칸의 2번째 페지

3 【Next】단추를 찰칵하여 Project Wizard대화칸의 다음 페지에 들어간다. 【Title】, 【Description】, 【Copyright】마당에 필요한 내용을 입력하고 【Finish】단추를 찰칵한다.

다음 아래에 제시한 순서에 따라 Applet조수를 사용해보자.

단계

1 【File】→【Next】지령을 선택하면 【Object Gallery】대화칸이 나타난다. 여기서 Web항목을 선택한 다음 오른쪽 부분에 있는 Applet아이콘을 두번 찰칵하거나 Applet아이콘을 선택한 상태에서 【OK】단추를 찰칵하여 Applet조수대화칸을 연다.(그림 5-4) 이 대화칸에서 추가선택항목 【Package:】는 기정으로 선택하고 【Class name:】본문칸에는 《MyUniversity》를 입력한다. 【Base class:】복합칸에서 《java.applet.Applet》를 선택한 다음에 검사칸【Generate header comments】, 【Can run standalone】과 【Generate standard methods】을 선택한다. 【Next】단추를 찰칵하여 다음 페지로 들어간다.

2 두번째 페지에서는 Applet에 파라메터를 추가하는 설정을 진행한다. Applet조수대화칸에서는 HTML파일의 Applet태그에 Param태그를 추가한 다음 원천코드에서 이 파라메터를 처리하는 코드를 삽입한다. Applet파라메터는 응용프로그람의 파라메터와 류사하다. 이 실레에서는 파라메터를 추가하지 않았다.



그림 5-4. Applet조수대화칸의 첫 페지

3 세번째 페지에서는 【Generate HTML Page】검사칸을 선택한다. 이 항목을 선택하면 체계는 자동적으로 Applet의 HTML파일을 호출한다. 【Title:】본문칸에 《MyUniversity》를 입력한다. 기타 다른 항목들은 기정으로 선택한다.(그림 5-5)



그림 5-5. Applet조수대화칸의 3번째 페지

4 Applet조수대화칸의 4번째 페지에서는 Applet배치를 진행한다. 사용자는 실행에 대한 배치를 하겠는가를 여기서 선택할수 있다. 만일 실행시 배치를 진행하려면 【Creat a runtime configuration】검사칸을 선택하면 된다. 이 실례에서는 이것을 선택할 필요가 없다.

5 【Finish】단추를 찰칵하여 Applet창조를 완성한다.

결과 프로젝트판에 2개의 새로운 파일 MyUniversity.java와 MyUniversity.html이 생긴다. 【File】→【Save All】지령으로 모든 내용을 보관한다.

이와 같이 Applet조수를 리용하여 대면부를 만들었다. 계속하여 대면부에 부품을 추가하는 실례를 보자.

6.4.2. Applet의 실행

Design태브를 찰칵하여 대면부설계창문을 연다. 구조판의 계층구조에서 this를 선택한다. 속성창문의 layout속성을 《null》로, background속성을 《Info》로 선택한다.(그림 5-6을 참고)

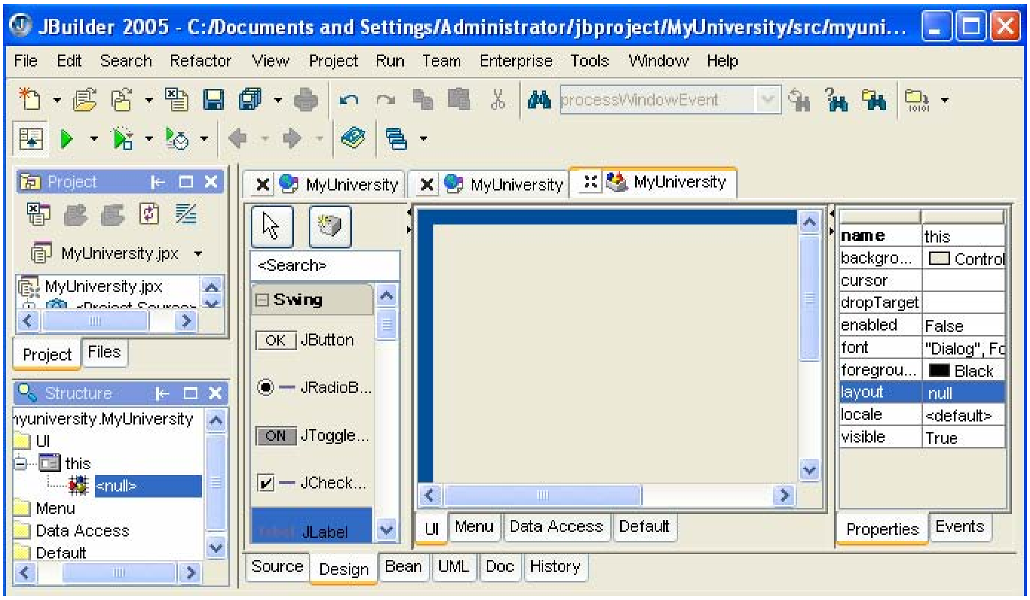


그림 5-6. Applet설계대면부

부품선택판에서 Swing서고의 jLabel을 찰칵하여 대면부에 5개의 jLabel부품을 추가한다. 이 부품들의 text속성을 각각 《대학》, 《학과》, 《소재지》, 《망주소》, 《보충정보》로 한다. foreground속성은 모두 《Blue》로 한다.

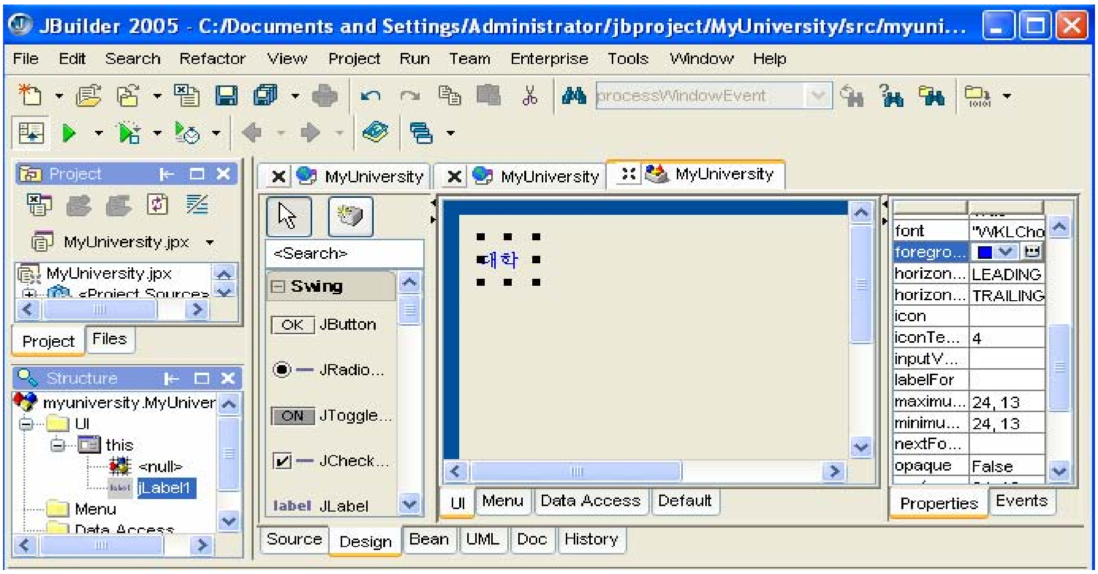


그림 5-7. Swing의 jLabel부품

다음으로 Swing의 jTextField부품을 3개 추가하고 text속성에서 기정값을 지운다. 하나는 jLabel1부품(대학)옆에 가져다놓고 사용자가 대학이름을 입력할 칸으로 리용하고 다른 하나는 jLabel4부품(망주소)옆에 가져다놓고 해당 대학의 망주소를 입력할 칸으로 리용한다. jTextField3은 jLabel5부품(보충정보)옆에 가져다놓고 사용자가 입력한 대학에대한 보충정보가 현시되도록 하는데 리용한다.

사용자가 학과이름을 선택할 때 리용할 Swing의 jComboBox1부품을 추가한다. 그다음 jComboBox부품인 jComboBox2을 《소재지》부품옆에 추가한다. 이것은 대학이 위치하고있는 소재지를 선택하는데 쓰인다. 이렇게 하여 설계된 대면부는 그림 5-8과 같다.



그림 5-8. 설계된 Applet대면부

계속하여 Swing의 jTextArea부품을 추가한다. 이때 text속성은 빈것으로, foreground속성은 Pink로 설정한다. 이것은 사용자가 입력한 정보를 현시하는데 리용된다. 다음 2개의 jButton부품인 jButton1과 jButton2를 추가하여 그것들의 text속성들을 각각 《확인》과 《재설정》으로 설정한다. 만일 대학에 대한 정보입력칸이 비여있으면 【확인】단추는 비능동상태에 있으며 사용자가 입력칸에 입력하면 이 단추는 능동상태로 된다. 사용자가 【확인】단추를 찰칵하면 프로그람은 입력한 내용과 선택된 정보를 jTextArea1에 현시한다. 만일 사용자가 다시 입력하려 한다면 【재설정】단추를 찰칵하여 이미 입력한 자료를 지운다.(그림 5-9)

다음으로 3개의 jCheckBox부품 jCheckBox1, jCheckBox2, jCheckBox3을 추가하고 text속성을 각각 《중요학과》,《3명이상의 박사》,《2명이상의 원사》로 설정한다. Background속성은 《orange》로 설정한다.(그림 5-10)



그림 5-9. 설계후 대면부(1)



그림 5-10. 설계후의 대면부(2)

마지막으로 【File】→【Save All】지령을 선택하여 보관한다.

아래에서 jComboBox부품사건에 대한 코드를 추가한다. 아래의 코드들은 복합칸에 학과이름과 대학의 소재지를 추가하는 부분이다. 처음에 프로그람을 실행할 때 jTextField

의 text속성이 비여있었으므로 【확인】단추는 비능동상태에 있게 된다. Source태브를 찰칵하여 원천코드창을 열고 init메쏘드를 선택한다.

이 메쏘드에 아래의 코드를 입력한다.

```
jComboBox1.addItem("정보통신학과");
jComboBox1.addItem("기계전자학과");
jComboBox1.addItem("자동화공학과");
jComboBox2.addItem("평양시 중구역");
jComboBox2.addItem("평양시 대동강구역");
jComboBox2.addItem("남포시 항구구역");
jComboBox2.addItem("함흥시 회상구역");
    jButton1.setEnabled(false);
```

JBuilder에는 코드편집시 편리를 위하여 클라스의 성원마당과 메쏘드들을 보여주는 기능이 있다.(그림 5-12) 실례로 사용자가 코드창문에 jTextArea1를 입력하고 점(.)을 찍으면 그 변수에 대한 메쏘드들과 마당들을 현시하는 창문이 나타나는데 여기서 사용자는 필요한것을 선택할수 있다.

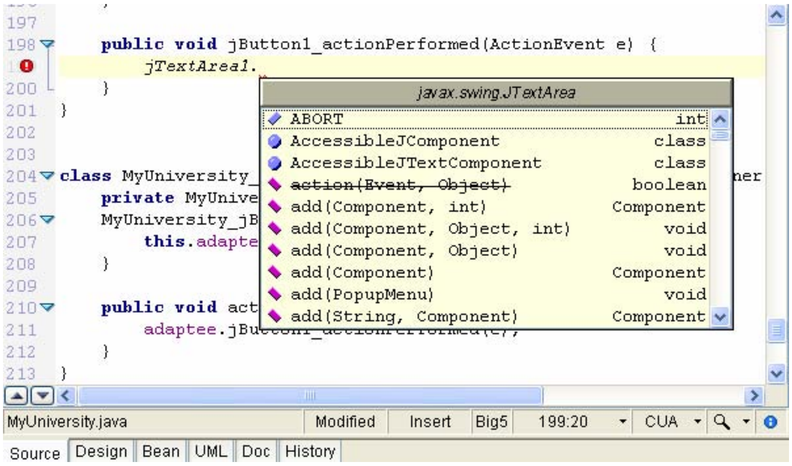


그림 5-12. 코드미리보기기능

매 부품들에 사건을 추가하는 방법은 대면부설계창문에서 부품(jComboBox1)을 선택하고 Event태브를 찰칵하여 필요한 사건을 선택하는것이다.

실례에서 필요한 사건은 itemStateChanged사건이므로 이것을 두번 찰칵하면 JBuilder는 자동적으로 상응한 메쏘드인 void jComboBox1_itemStateChanged(ItemEvent e)를 생성한다. 여기에 아래와 같은 코드를 입력하면 된다.

```
    jTextArea1.setText( “” );
    jTextArea1.setForeground(Color.black);
    jTextArea1.append( “우리 대학을 소개합니다.” );
```

jButton1의 actionPerformed사건을 리용하는것은 TextArea1에 사용자가 입력한 각종 정보를 현시하기 위해서이다. 코드내용은 아래와 같다.

```
      void button1_actionPerformed(ActionEvent e) {
    textArea1.setForeground(new Color(255,0,0));
    textArea1.setText("");
    if (textField1.getText()!=""){
      textArea1.append("대학이름:"+textField1.getText()+"\r\n");
      textArea1.append("대학의 소재지:"+jComboBox2.getSelectedItem()+
"\r\n");
      if(jComboBox1.getSelectedItem()=="정보통신학과"){
        textArea1.append("학과이름:정보통신학과\r\n");
      }
      else if(jComboBox1.getSelectedItem()=="기계전자학과"){
        textArea1.append("학과이름:기계전자학과\r\n");
      }
      else if(jComboBox1.getSelectedItem()=="자동화공학과"){
        textArea1.append("학과이름:자동화공학과\r\n");
      }
      if(jCheckBox1.isEnabled()==true){
        textArea1.append("이 학과는 중점학과입니다.\r\n");
      }
      if (jCheckBox2.isEnabled()==true){
        textArea1.append("이 학과에는 3명이상의 박사가 있습니다.\r\n");
      }
      if (jCheckBox3.isEnabled()==true){
        textArea1.append("이 학과에는 2명이상의 원사가 있습니다.\r\n");
      }
      textArea1.append("대학의 망주소:"+textField2.getText()+"\r\n");
      textArea1.append("추가정보:"+textField3.getText()+"\r\n");
    }
```

【File】→【Save All】지령으로 작성된 모든 코드들을 보관한다.

다음으로 Applet를 실행한다. 프로젝트판의 MyUnivesity.html을 선택한다.【Run】→【Run Project】지령을 찰칵하여 실행하면 그림 5-13, 그림 5-14와 같은 결과를 얻는다.

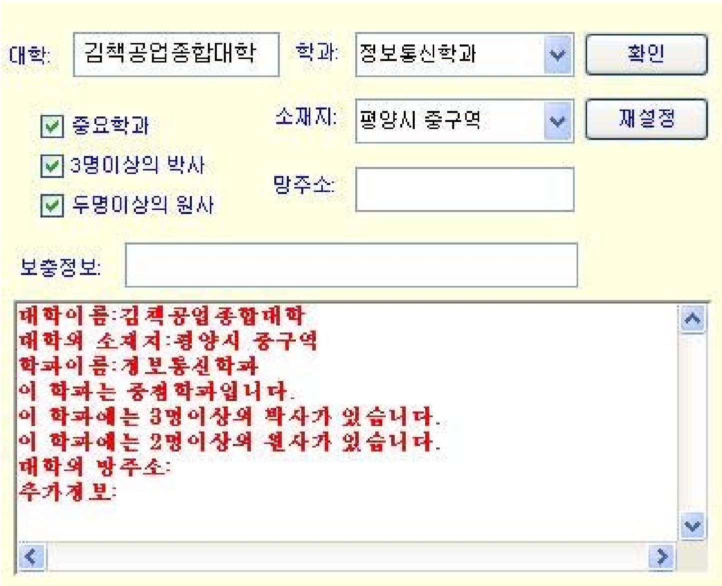


그림 5-13. 프로그람의 실행결과1



그림 5-14. 프로그람의 실행결과2

6.4.3. JBuilder에서 Applet배비

Applet배비란 Java클라스파일, 화상파일, 음성파일과 기타 Applet에 필요한 원천파일들을 하나의 파일로 묶음으로써 HTTP로 열람할 때 하나의 파일로 내리적재할수 있도록 한다는것을 말한다. JAR는 Java Archive의 간략어이다. JAR도구의 기본기능은 파일을 압축하여 묶는것이다.

JAR의 특징은 아래와 같다.

- 가동기반에 무관계하며 계승성이 좋은것이다.
- 여러개의 Java프로그람들을 하나로 묶음으로써 HTTP에서 내리적재속도를 높여준다.
- Applet작성자에게 인증을 제공한다.
- JBuilder의 Archive Builder는 Applet에 필요한 모든 원천파일들을 하나의 JAR 파일에 묶어 주어 작업량을 줄인다.

구체적인 방법은 다음과 같다.

단계

1 【File】→【New】→【Archive】지령을 선택하여 Archive조수대화칸을 펼친다.(그림 5-15) 여기에서 AppletJAR아이콘을 선택하고 【OK】단추를 찰칵한다.

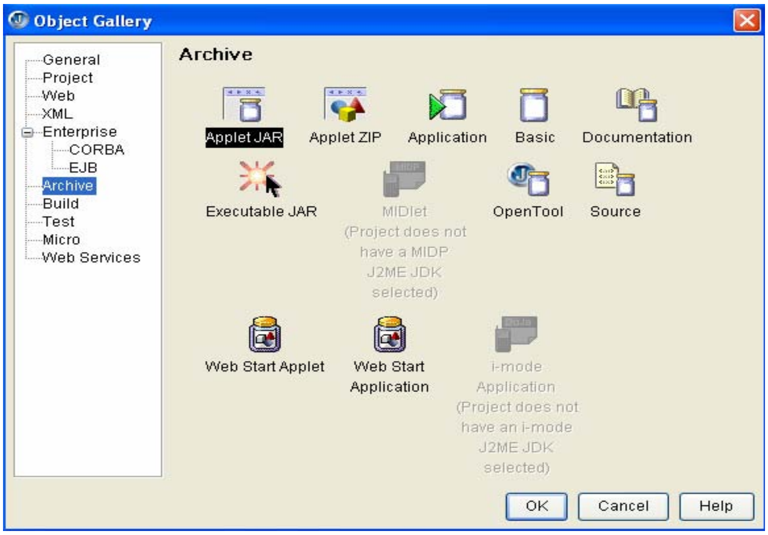


그림 5-15. Archive의 대면부



그림 5-16. Archive Builder조수대화칸의 첫 페지

2 Archive Builder조수대화칸의 첫 페지에서 【Name:】본문칸에 사용자의 요구에 맞게 이름을 입력한다. 또한 경로도 지정한다.(그림 5-16)

3 【Next】단추를 찰칵하여 2번째 페지에 들어간다. 여기에서 필요한 설정을 진행한다.(그림 5-17)

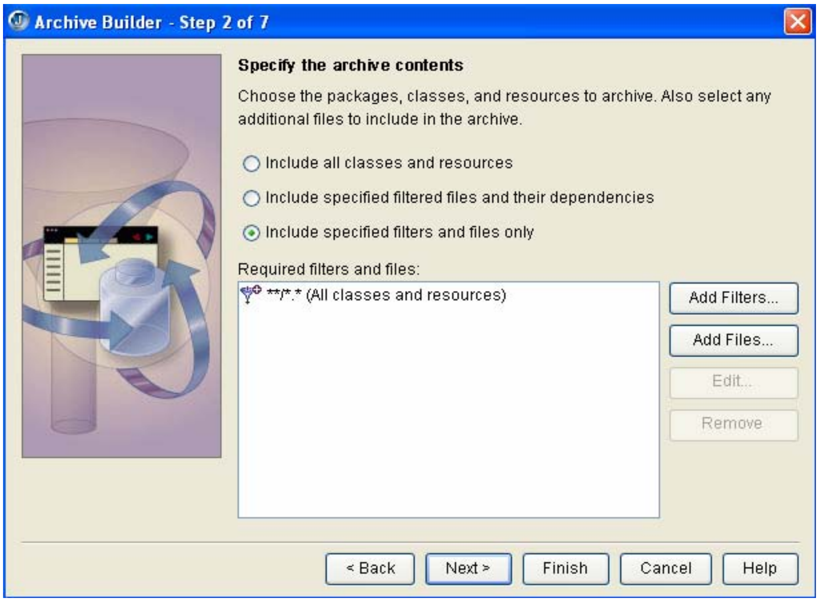


그림 5-17. Archive Builder조수대화칸의 2번째 페지

4 기타 추가선택항목들을 기정으로 선택하고 【Finish】단추를 찰칵하여 Archive Builder조수대화칸을 닫는다. 이때 Applet JAR가 왼쪽 프로젝트판에 나타난다.

5 계속하여【Project】→【Make Project】지령으로 프로젝트를 콤파일한다. Archive Builder는 프로젝트등록부의 모든 파일들을 JAR파일에 넣는다.

6 MyUniversity.html에서 Archive속성에 MyUniversity.jar파일을 추가한다.

```
<Applet
codebase= “.”
Code= “University.MyUniversity.class”
Archive=” MyUniversity.jar”
Name= “Test Applet”
WIDTH=” 500”
height= “300”
hspace= “0”
vspace= “0”
align= “middle”
>
</Applet>
```

마지막으로 Applet의 HTML파일과 JAR파일을 봉사기의 해당한 위치에 복사하면 된다.

5.4.4. JBuilder에서 배비한 Applet의 원천코드

MyUniversity.html의 원천코드는 아래와 같다

```
<html>
<head>
<meta http-eguivz= “Content-Type” content= “text/html;charset= Big5”
>
<title>
HTML Test Page ..........    My University
</title>
</head>
<body>
University.MyUniversity will appear below in a java enabled browser.
 <br>
<Applet
codebase= “.”
```

```
Code= "University.MyUniversity.class"
Archive= "MyUniversity.jar"
Name= "TestApplet"
WIDTH= "500"
height= "300"
hspace= "0"
vspace= "0"
align= "middle"
>
</Applet>
</body>
</html>
```

여기서 MyUniversity.java의 원천코드는 략한다.

제5절. Applet의 시험

5.5.1. 시험의 기본순서

Applet의 시험순서는 아래와 같다.

단계

1 Applet의 파일배비에서 주의할 점은 다음과 같다.

- Applet파일의 경로는 반드시 Applet태그(<Applet>)의 codebase속성에 부합되여야 한다.
- 대소문자를 정확히 구별하여 써야 한다.
- 등록부이름은 반드시 패키지이름과 같아야 한다.
- Archive속성에 주의하여야 한다.

2 지령행창문을 펼친다.

3 <Applet>태그의 classpath의 변수값을 버린다. 그리고 AppletView가 실행하려는 Java파일의 위치를 알수 있도록 해준다.

4 HTML파일과 JAR파일이 있는 등록부들을 변경시킨다.

실레로

<JB>/<jdk>/AppletViewer MyUniversity.html

여기에서 <JB>는 JBuilder가 설치된 경로이고 <jdk>는 JDK의 등록부경로이다.

5.5.2. 열람기에서의 시험

서로 다른 열람기들에서 Applet에 필요한 각종 클라스파일과 원천파일들이 정상적으로 실행되는가를 시험해본다. 동시에 열람기에서 Applet를 실험하면서 HTML에 삽입해 넣은 각종 파라메터들의 값들, 실례로 파일경로, 각종 필요한 원천파일들이 정확한가를 확인한다.

열람기에서 Applet를 시험할 때 아래와 같은 내용들에 주의를 돌려야 한다.

- 열람기가 JDK를 지원하는가.
- 열람기의 Java조종탁에서 오유가 없는가.

Applet를 재콤파일한 후 열람기의 고속완충기억기(Cache)에 보관되여있는 낡은 내용들을 지우고 다시 열람한다.

제6장. JSP프로그람작성

JSP(Java Serve Pages)는 Java가동기반에서 실행되며 J2EE환경에서 동적페지프로그람을 제공하는 기술이다. JSP는 Servlet에 기초한 봉사기측 Java동적웨브페지작성스크립트언어이다. HTML과 같이 정적페지를 제작할수 있을뿐아니라 동적페지도 작성할수 있다.

JSP에 의한 의뢰기와 봉사기사이의 접속과정을 그림 6-1에서 보여준다.

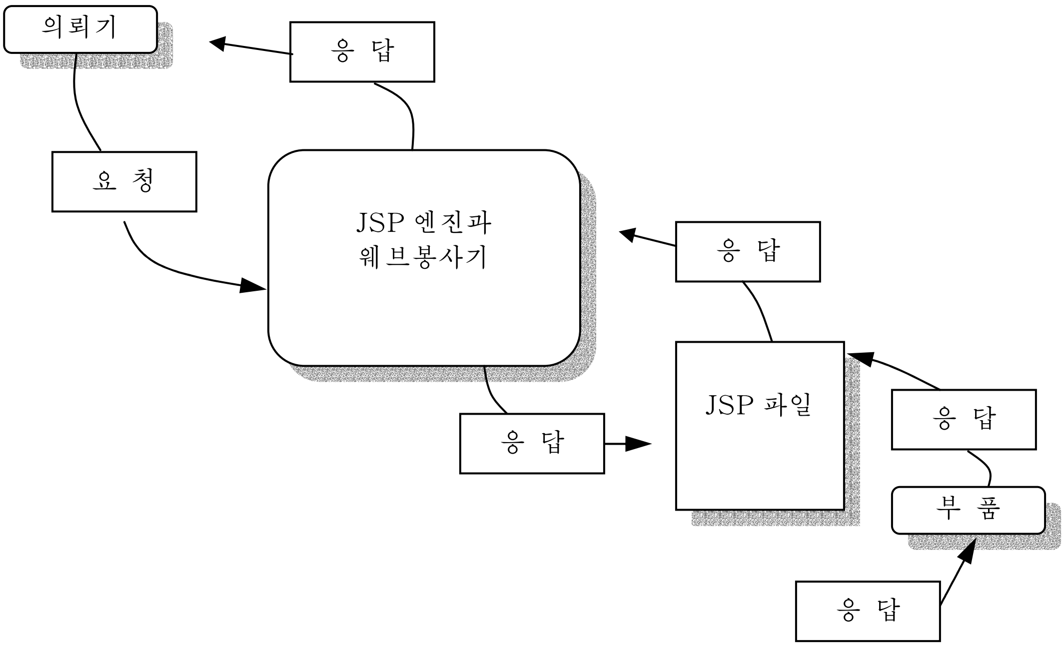


그림 6-1. JSP의뢰기와 봉사기사이의 접속과정

제1절. JSP와 웨브

JSP기술은 동적웨브페지를 간결하면서도 빨리 작성하는 방법을 제공하고있다. 의뢰기측이 열람기를 통하여 웨브홈페지를 방문하려고 할 때 웨브봉사기는 의뢰기의 열람기에 웨브파일을 내리적재해준다.

웨브페지에는 기본 3가지 류형이 있다.

-정적페지

이 페지는 웨브봉사기에 보관되며 개발자들이 프로그람을 작성하였을 때 이미 정해져 내용이 변하지 않는다. 즉 접근할 때마다 내용이 변하지 않는다. 이 형식의 우점은 프로그람작성이 쉽고 간단하며 믿음성이 높은것이다. 결함은 능동적이지 못하고 호상자료교환기능이 부족한것이다. 또한 내용이 변하면 반드시 다시 설계하여야 한다.

-동적페지

이 페지는 의뢰기의 열람기가 웨브홈페지를 방문할 때 웨브봉사기에 의해 만들어진다. 의뢰기의 열람기가 웨브봉사기에 요청하면 웨브봉사기는 그에 맞는 프로그람을 실행하여 그 결과를 HTML형식으로 열람기에 되돌려준다. 동적페지의 우점은 능동성과 호상작용이 강한것이다. 결함은 정적페지보다 속도가 뜬것이다. 또한 최종결과가 의뢰기에 전달된 후에도 내용이 변하지 않는것이다.

-활동성페지

이것은 정보원천에 직접 접근할수 있게 하며 련속적으로 내용을 변화시킬수 있다. 활동성페지에서는 동적페지에서 갱신하지 못한 결함들을 극복하고있다. 그러나 이것 역시 속도가 상대적으로 느린 결함을 가지고있다.

다음으로 몇가지 동적페지기술에 대하여 소개한다.

(1) CGI

초기의 홈페지들은 정적방법을 리용하였는데 망응용이 급속히 발전하면서 사용자들의 요구를 만족시킬수 없었다. 사용자들 호상간의 대화를 보장하고 자료기지를 관리조종하는 등 봉사를 할수 있는 동적망이 필요하게 되였다. 이러한 문제들에 대한 초기해결방안으로 CGI(Common Gateway Interface:공통관문대면부)를 리용하였다. CGI를 실현한 웨브봉사기의 역할은 봉사기가 사용자의 요구와 요구하는 자료사이의 관문(gateway)으로 되는것이였다. 의뢰기가 웨브봉사기에 요청하면 봉사기는 의뢰기측의 요청에 따라 하나의 프로쎄스를 만들어 CGI프로그람을 실행시킨다. 그러나 CGI방식의 가장 큰 결함은 효률이 낮고 신축성이 없는것이다. 웨브봉사기가 요청을 받을 때마다 새로운 프로쎄스를 만들어야 하는데 요청수가 많은 경우에는 웨브봉사기의 자원이 없어져 봉사기가 폭주될수 있는 가능성이 많

다. CGI의 결함을 극복하기 위하여 전문적인 웨브봉사기 API 즉 ISAPI와 NSAPI가 나왔다. 프로그람작성자들은 서고들을 리용하여 프로그람을 작성하였지만 여기에도 결함이 있었다. 그것은 특정한 가동기반에 국한되고 프로그람서고를 많은 사용자가 동시에 방문할 때 보안이 보장되여야 하였다.

(2) ASP(Active Server Page)는 마이크로쏘프트회사가 개발한것으로서 동적웨브페지를 설계할수 있게 하는 기술이다. 이것의 우점은 HTML태그를 포함할수 있으며 자료기지에 직접 접근하고 Activex조종부품을 무한히 확장할수 있는것이다. 결함은 Microsoft Windows체계에서만 사용할수 있다는것이다.

(3) PHP(Personal Home Page)는 Linux봉사기에서 광범히 활용되고있는 언어로서 가볍고 빠르며 원가가 적게 드는 우점을 가지고있다. 결함은 반드시 Apache와 함께 사용해야 한다는것이다.

우에서 설명한것과 같이 해결해야 할 문제들은 다음과 같다.

- 임의의 웨브 혹은 응용프로그람봉사기에서 실행시킬수 있어야 한다.
- 응용프로그람과 페지현시를 분리시켜야 한다.
- 개발 및 시험속도가 빨라야 한다.
- 개발과정이 간단하여야 한다.

(4) JSP(Java Server Pages)는 이러한 요구들을 만족시킨다. JSP는 Java언어에 기초하고있는것으로 하여 Java와 류사한 특징을 가지고있다. 따라서 한번 작성하여 아무데서나 실행시킬수 있다. JSP는 개발, 이식, 실행방법에서 우월하며 구체적인 장치적가동기반이나 조작체계, 봉사기에 의존하지 않는다. 또한 여러 개발회사들이 제공하는 각종 도구들을 지원한다.

가동기반에 무관계한 Java Bean부품, Enterprise Java Beans부품(EJB), 태그서고(Taglibrary) 등의 부품들을 서로 다른 개발자들이 반복리용할수 있다.

정적페지와 동적페지내용의 개발을 서로 분리시킬수 있다. 이것은 동적내용을 고치지 않고도 망페지의 정적효과(례를 들어 글자의 색갈 크기 등)들을 수정할수 있다. 또한 부품의 대면부가 변하지 않는한 동적인 내용을 다시 작성하여도 망페지의 효과는 영향을 받지 않는다.

JSP는 자료기지와 련결할 때 JDBC기술을 리용한다. 이때 망페지는 JDBC구동프로그람을 통하여 자료기지와 련결된다. 현재 대다수의 자료기지체계는 ODBC구동프로그람을 가지고있다. Sun회사는 JDBC와 ODBC사이의 련결기술을 개발하여 ODBC구동프로그람을 가지고있는 자료기지를 방문할수 있게 하였다.

제2절. JSP의 API참조

여기에서는 JSP API를 JSP1.1에 기초하여 간단히 소개한다.

아래에서 JSP의 기본태그들을 소개한다. JSP는 동적망페지를 설계하기 위한 웨브언어이므로 웨브언어형식의 태그들을 사용한다. 표 6-1에서 JSP에서 비교적 중요한 태그들을 주었다.

표 6-1. JSP의 상용태그

류 형	기 능
<%코드%>	Java언어로 프로그람을 작성할 때 리용하는 태그이다.
<%----JSP해석----%>	JSP태그에 대한 해석
<%=변수 혹은 메쏘드의 값%>	결과를 계산하고 그것을 문자렬로 절환한다. 이것은 출구페지에 직접 반영된다.
<%!선언%>	망페지에서 리용되는 변수 혹은 메쏘드를 선언한다.
<jsp:use Bean>	리용하는 빈의 이름, 류형, 생명주기 등을 정의한다.
<%@page page_directive_attr_list%>	이 페지에서 리용하는 패키지를 설치한다.

JSP를 리용하여 동적망페지를 개발하는데는 크게 2가지 방법이 있다.

첫째 방법은 HTML페지에 직접 Java코드를 삽입하는것으로서 이러한 방법은 Servlet의 원천코드를 직접 HTML코드에 삽입하는것과 같다. 이 방법은 개발자가 HTML과 Java언어를 잘 아는 경우에 리용한다. 이 방법의 우점은 관리해야 할 파일수가 비교적 적다는것이고 결함은 코드를 리해하기가 힘들고 망페지구조가 복잡한것이다.

둘째 방법은 JSP페지에서 JavaBean를 직접 호출하는것이다. 이 방법은 HTML코드와 Java코드를 분리시킨것으로 하여 구조가 간결하고 명백하며 리해하기가 비교적 쉽다. 뿐만아니라 부품들을 반복리용할수 있는것으로 하여 큰규모의 홈페지개발에 적합하다.

JBulider는 JSP파일창조를 위한 JSP조수, JSP태그의 Code Insight기술 등을 가지고있다. JBuilder가 제공하는 JSP조수를 리용하는 실례를 고찰하자.

실례 6-1(JSP 조수를 리용한 프로그람작성)

단계

1 【File】-【New Project】지령을 찰칵한다.(그림 6-2) 프로젝트이름을 Showtime 라고 입력하고 나머지는 기정으로 설정한다.

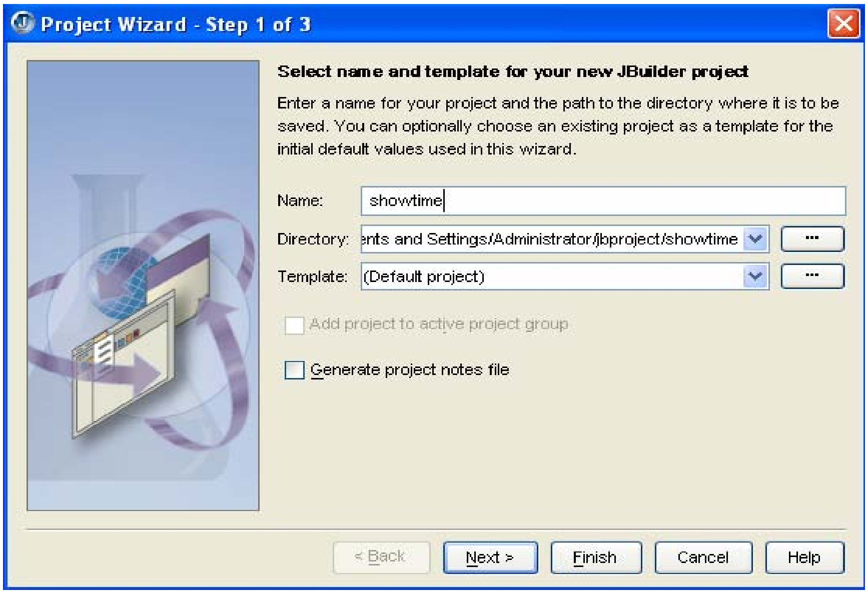


그림 6-2. showtime프로젝트창조

2 【Next】단추를 찰칵하여 두번째 페지로 들어간다. 여기에서는 모든 추가선택항목을 기정으로 설정한다.

3 【Next】단추를 찰칵하여 세번째 페지로 들어간다. 【Title:】마당에 《TestJSP》라고 입력하고 기타 추가선택항목들인 Description과 Copyright는 필요에 따라 써넣는다.(그림 6-3)

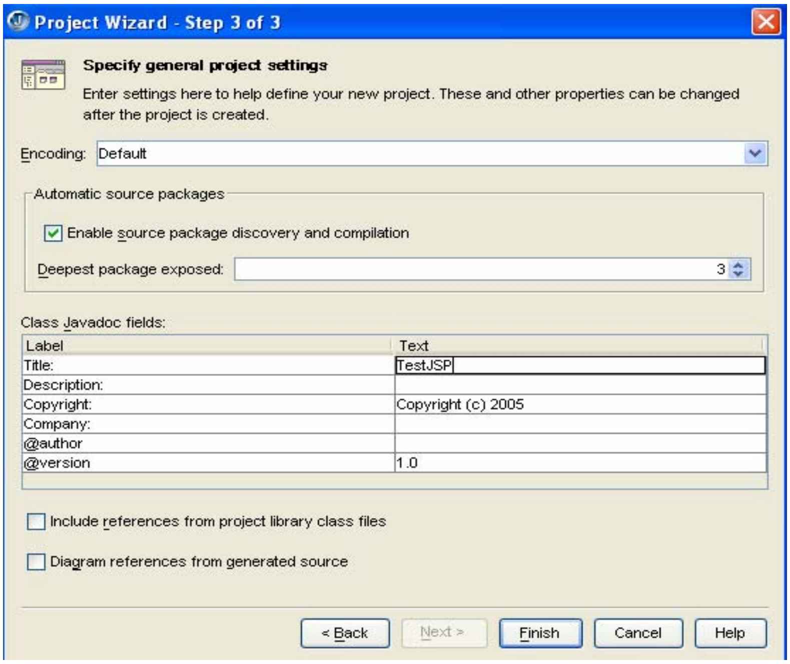


그림 6-3. Project Wizard대화칸의 3번째 페지

4 【Finish】단추를 찰칵하면 프로젝트파일과 서술파일이 만들어지는데 그것들의 이름은 각각 Showtime.jpx와 Showtime.html이다.

웨브응용프로그람은 사용자의 모든 망페지내용들을 포함하고있는 하나의 등록부나무로서 실행에 필요한 모든 자원들을 포함하고있다. 여기에 web.xml파일이 있는데 이것은 다른 사람이 사용자의 웨브페지에 접근하려고할 때 필수적으로 웨브봉사기에 주어야 하는 정보이다.

아래에서 웨브응용프로그람을 창조하는 과정을 구체적으로 서술한다.

단계

1 【File】→【New】지령을 선택하면 【Object Gallery】대화칸이 나타난다. 여기서 Web항목을 선택한 다음 오른쪽 부분에 있는 Web Module(WAR)아이콘을 두번 찰칵하거나 Web Module(WAR)아이콘을 선택한 상태에서 【OK】단추를 찰칵한다.(그림 6-4)

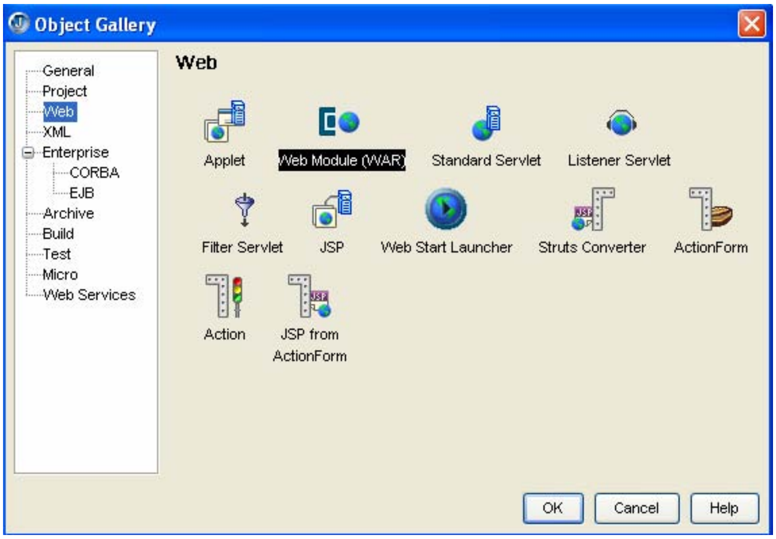


그림 6-4. Web의 대면부

2 이때 나타나는 【Web Module Wizard】대화칸에서 웨브모듈의 이름과 경로를 입력한다.(그림 6-5, 6-6)

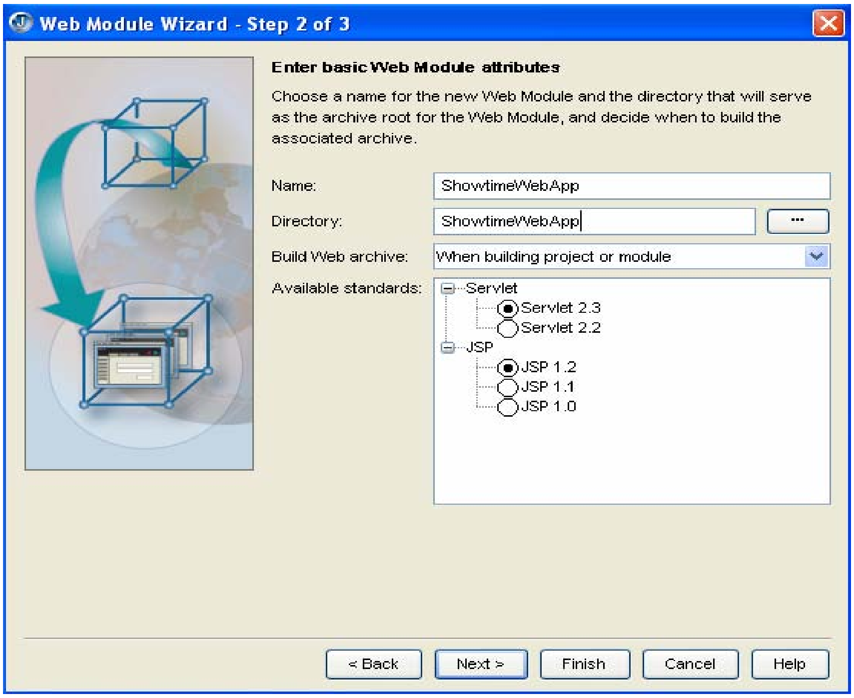


그림 6-5. Web Module조수대화칸의 2번째 페지

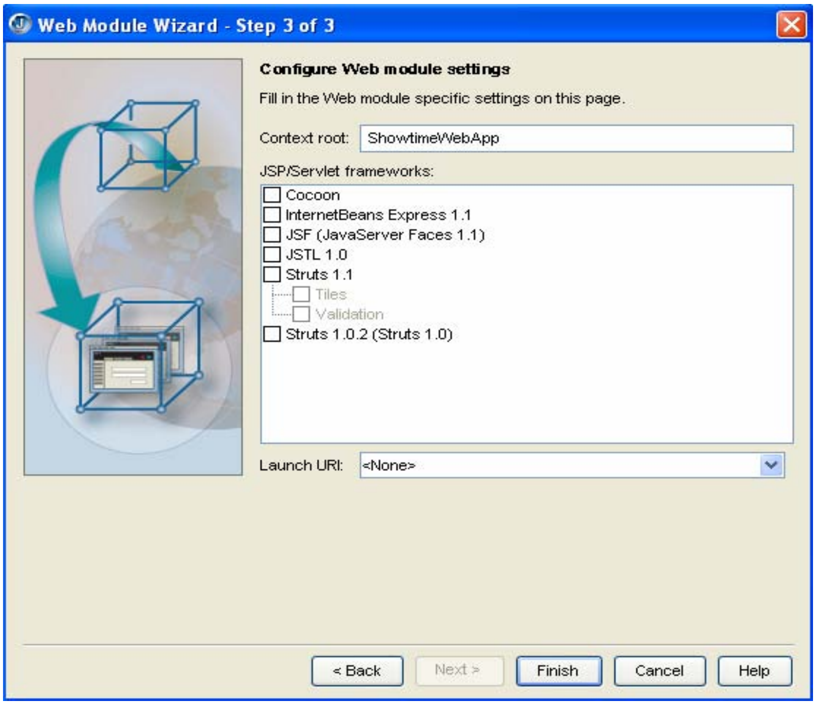


그림 6-6. Web Module조수대화칸의 3번째 페지

3 【OK】단추를 찰칵하면 ShowtimeWeb App가 생긴다.

4 【File】→【New】지령을 선택하여 【Object Gallery】대화칸을 열고 Web항목을 선택한다. 오른쪽에 있는 JSP아이콘을 선택한다.(그림 6-7)

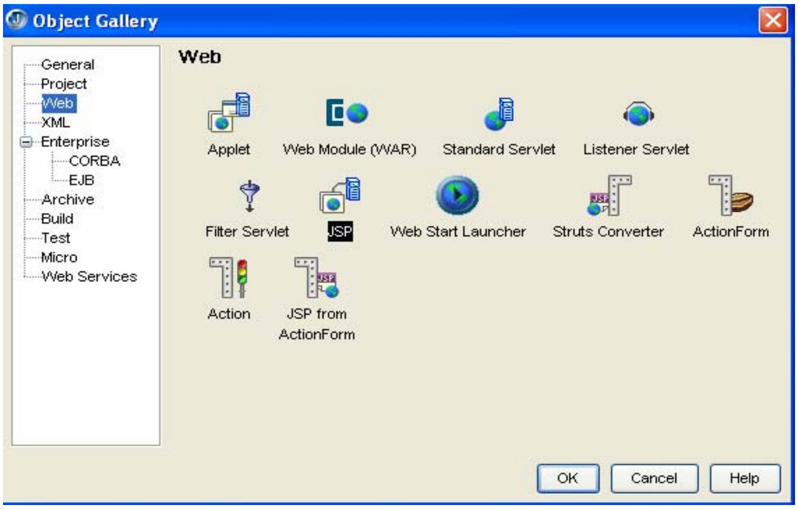


그림 6-7. Object Gallery대화칸의 Web페지

5 【OK】단추를 찰칵하여 JSP Wizard대화칸의 첫 페지에 들어간다. JSP파일의 이름을 ShowtimeJSP로 고친다.(그림 6-8)



그림 6-8. JSP Wizard대화칸의 첫 페지

6 【Next】단추를 찰칵하여 다음 페지로 들어간다. background를 연한 황색으로 선택한다.(그림 6-9)

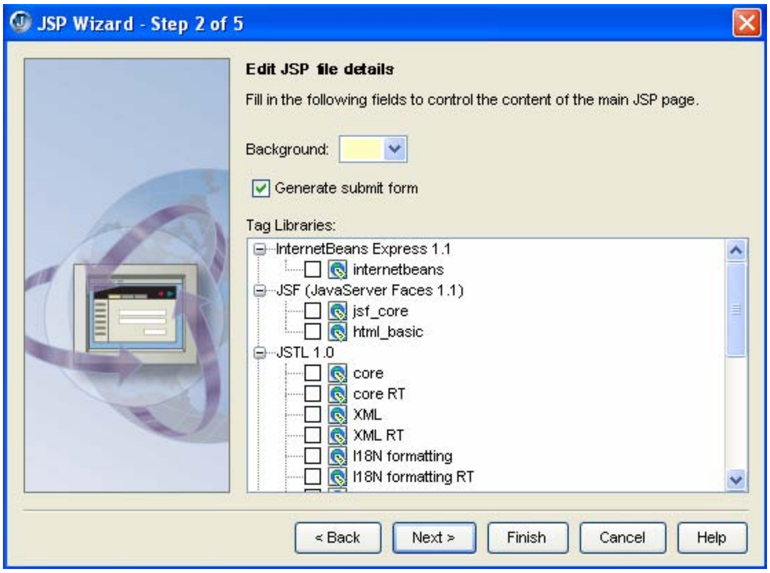


그림 6-9. JSP Wizard대화칸의 2번째 페지

7 【Next】단추를 찰각하여 다음 페지로 들어간다. 여기에서는 JSP페지에서 사용할 JavaBean을 설정하여야 하는데 기정으로 설정한다.(그림 6-10)

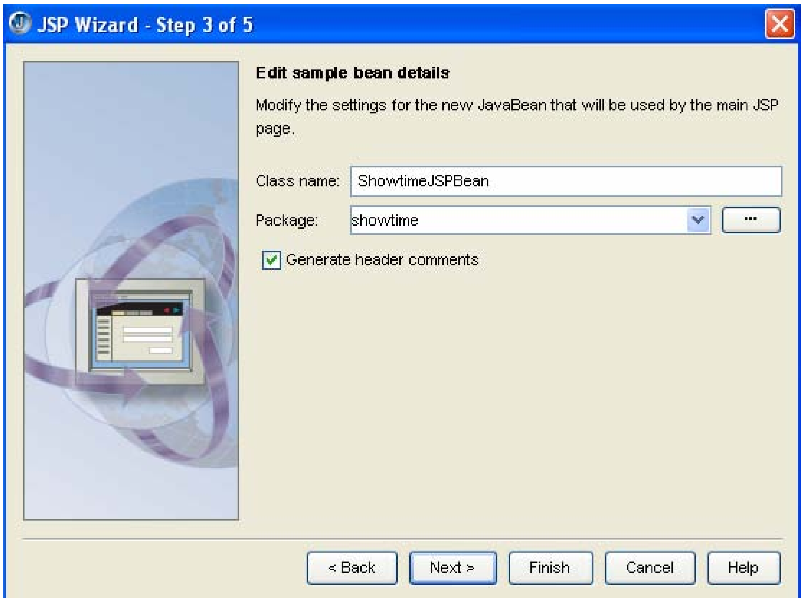


그림 6-10. JSP Wizard의 3번째 페지

8 【Next】단추를 찰각하여 다음 페지에 들어간다. 모든 추가선택항목을 기정으로 한다.(그림 6-11)

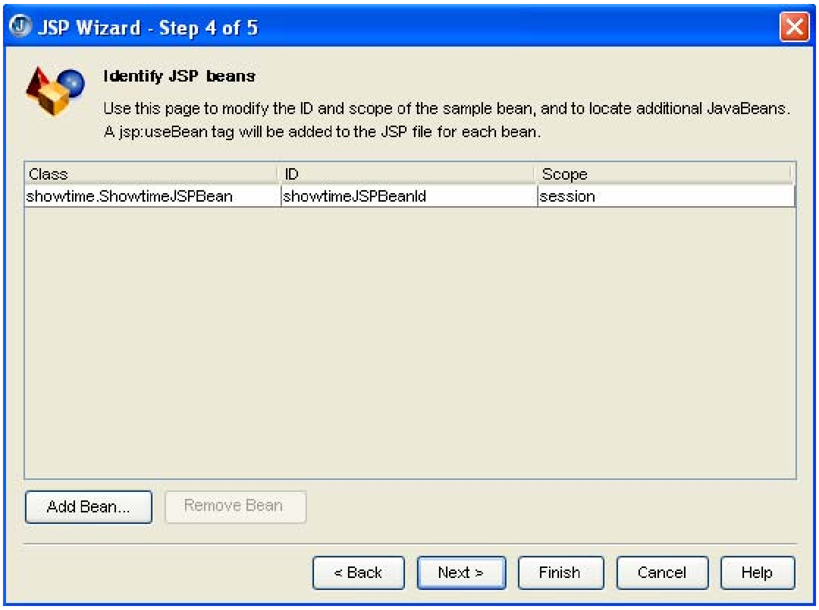


그림 6-11. JSP Wizard대화칸의 4번째 페지

9 【Next】단추를 찰칵하여 다음페지로 들어간다. 여기에서는 기본적으로 JSP배치항목을 설정한다. 【Create a runtime configunation】검사칸을 선택하면 새로운 JSP설정을 신속히 진행할수 있다. 【Base Configunation】복합칸에서 ShowtimeJSP를 선택한다.(그림 6-12)



그림 6-12. JSP Wizard의 5번째 페지

10 【Finish】단추를 찰칵하여 JSP조수대화칸을 닫는다. 프로젝트판에서 새로 만들어진 ShowtimeJSP.jsp를 볼수 있다.(그림 6-13) 자동생성된 ShowtimeJSP.jsp의 8행에 <jsp:useBean id="showtimeJSPBeanId" scope="session" class="showtime.ShowtimeJSPBean" />라는 내용이 있다. 여기에서는 JSP에서 리용하는 bean클라스를 지정해준다. bean클라스는 JSP를 창조할 때 지정해준것이다.(그림 6-10) 이 클라스도 JSP조수에 의하여 자동생성된다.

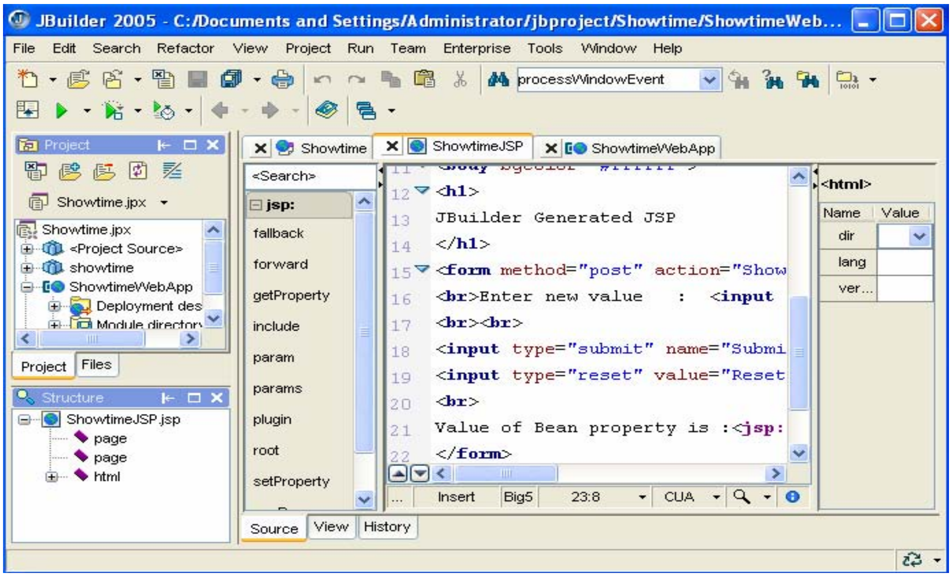


그림 6-13. 새로 생성된 ShowtimeJSP.jsp

다음으로 ShowtimeJSP.jsp에 코드를 입력하여 시간을 현시하게 하자. 먼저 프로젝트판에서 ShowtimeJSP.jsp를 선택하고 불필요한 코드를 지워버린다. 아래에 지워버려야 할 코드를 주었다.

```
<h1>
JBuilder Generated JSP
</h1>
<form method= “submit”  name= “Submit”  value= “Submit” >
<br><br>
<input type= “reset”  value= “Reset” >
</form>
```

다음 아래의 코드를 입력한다.

```
<h1>
<현재 Web봉사기의 시간>
</h1>
<% java.util. Date Mydate=new java.util.date();%>
<%=Mydate.getManth()+1%> 월
<% =Mydate.getDate()%> 일
<% =Mydate.getHours()%> 시
<% = Mydate.getMinutes()%> 분
```

코드를 입력한 다음의 ShowtimeJSP.jsp파일을 그림 6-14에서 보여준다.

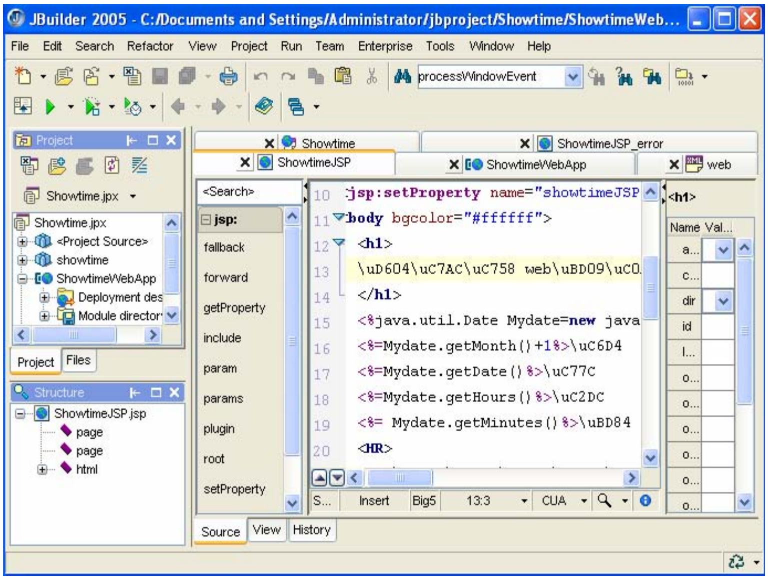


그림 6-14. ShowtimeJSP.jsp에 코드입력

아래에서는 JSP파일에서 정적페지의 삽입에 대하여 소개한다.

제3절. JSP의 HTML설계

ShowtimeJSP.jsp에 의뢰기의 시간을 현시하는 HTML코드를 입력하고 선택단추를 삽입하여 사용자가 의뢰기와 봉사기의 시간을 비교하고 그 결과를 되돌려 보내는 프로그람을 작성해보자.

ShowtimeJSP.jsp에 아래의 코드를 입력한다.

```
<hr>
<h1> 의뢰기시간 현시 </h1>
<script Language = ‘javaScript’ >
van clientDate = new Date();
document. White (ClientDate.get Yean()+ “년”);
document. White (ClientDate.get Month()+ “월”);
document. White (ClientDate.get Date()+ “일”);
document. White (ClientDate.get Hours()+ “시”);
document. White (ClientDate.get Minutes()+ “분”);
</script>
</hr>
```

계속하여 ShowtimeJSP.jsp에 아래의 코드를 입력한다. 이것은 망페지에 2개의 선택단추를 삽입하여 봉사기와 의뢰기시간이 일치하는가 일치하지 않는가를 사용자가 판단하게 한다.(그림 6-15)

```
<hr>
<p> 봉사기와 의뢰기의 시간이 일치하는가를 판단</p>
<FORM ACTION = “Showtimejsp2.jsp” METHOD = “POST”>
<INPUT TYPE = “radio” NAME = “Showtimejsp” VALUE = “Is Sa
me.”> 일치합니다. <br>
<INPUT TYPE = “radio” NAME = “Showtimejsp” VALUE = “Isn’t S
ame.”> 일치하지 않습니다. <br>
<INPUT TYPE = “submit” NAME = “submit” VALUE = “확인”>
</FORM>
```

우의 코드삽입결과는 그림 6-15에서 보여준다.

사용자의 판단결과를 현시하기 위하여 또 다른 JSP를 만들어야 한다.

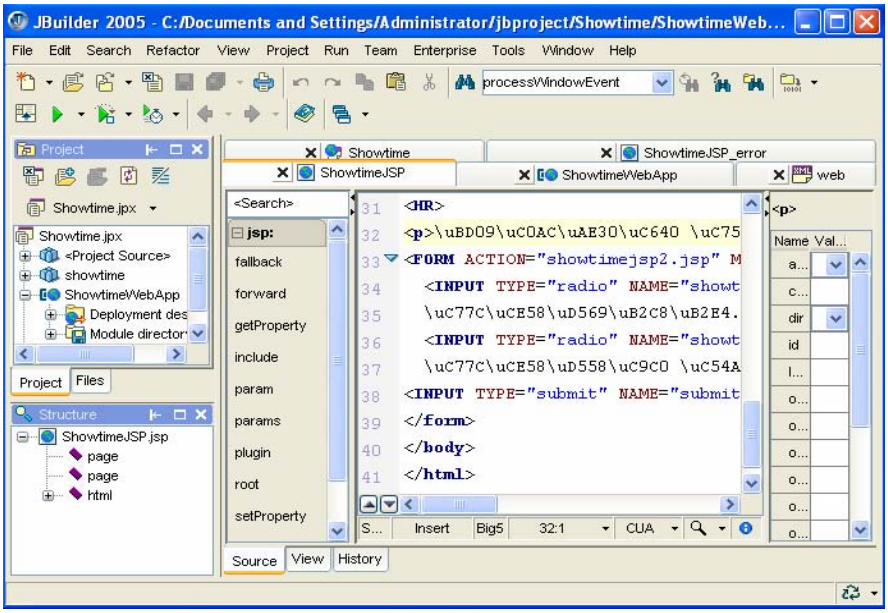


그림 6-15. Showtimejsp.jsp에 HTML코드의 삽입

실례 6-2(사용자판단결과를 현시하기 위한 프로그람작성)

구체적인 순서는 다음과 같다.

단계

1 【File】→【New】지령을 선택하고 Web항목을 찰칵한다. 여기서 JSP아이콘을 선택한 다음 【OK】단추를 찰칵하면 JSP Wizard대화칸이 펼쳐진다.(그림 6-16)

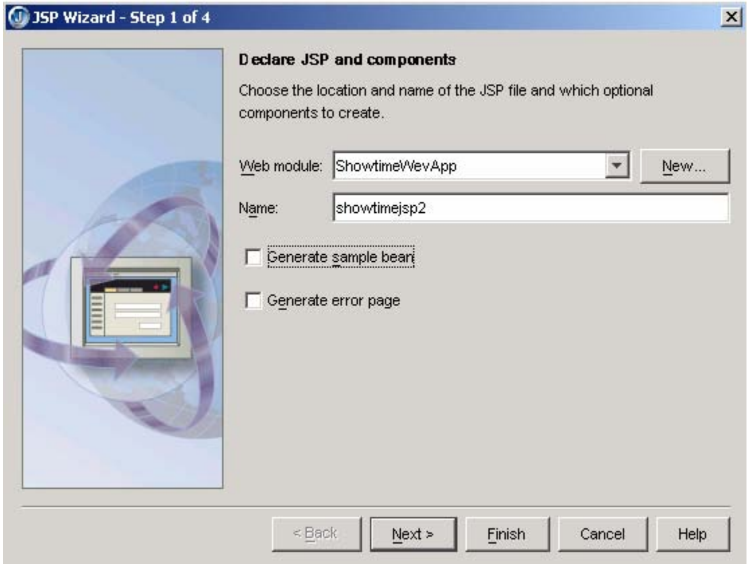


그림 6-16. JSP Wizard대화칸의 첫 페지

2 추가선택항목들을 기정으로 설정하고 【Finish】단추를 찰칵하여 JSP조수를 완성한다. 이때 Showtimejsp2.jsp파일이 생기는데 여기에서 불필요한 코드를 삭제한다. 실례로

```
<h1>
JBuilder Generated JSP
</h1>
```

다음 아래의 코드를 삽입한다. 이 코드들은 Showtimejsp.jsp에서 보내는 정보를 현시한다.

```
<%@ Page ContentType = "text/html;Charset = Big5" %>
<html>
<head>
<title>
Showtimejsp.jsp에서 온 정보
</title>
</head>
```

```
<body bgcolor = "#ff0000">
봉사기와 의뢰기시간이 일치합니까? <BR>
봉사기와 의뢰기의 현재시간 :
"<%=reguest.getParameter ("Showtimejsp")%>"
<BR>
</body>
</html>
```

이것으로 정적 HTML설계는 완성되였다. 설계한 실례를 실행시켜 보자.

제4절. JSP의 실행

사용자는 프로그람을 JBuilder에서 실행시킬수도 있고 열람기에서 직접 실행시킬수도 있다. 그러나 열람기에서 실행하려면 Java실행환경이 있어야 하며 Tomcat봉사기를 설치하여야 한다. 여기에서는 JBuilder에서의 JSP실행만을 소개한다.

6.4.1. JSP파일의 콤파일

다른 파일들과 마찬가지로 JSP파일도 콤파일하여야 한다. 구체적인 순서는 아래와 같다.

【Project】→【Make Project】지령으로 프로젝트를 콤파일하거나【Rebuild Project】지령으로 콤파일을 진행한다.(그림 6-17)

콤파일할 때 JSP파일하나만을 선택하여 할수 있다. 실례로 Showtimejsp2.jsp만을 콤파일하려면 Showtimejsp2.jsp를 선택하여 편집창문에 현시한 후 프로젝트판의 프로젝트계층구조에서 마우스오른쪽단추를 찰칵한다. 이때 나오는 지름차림표에서 【Make】혹은【Rebuild】지령을 찰칵한다.(그림 6-18)

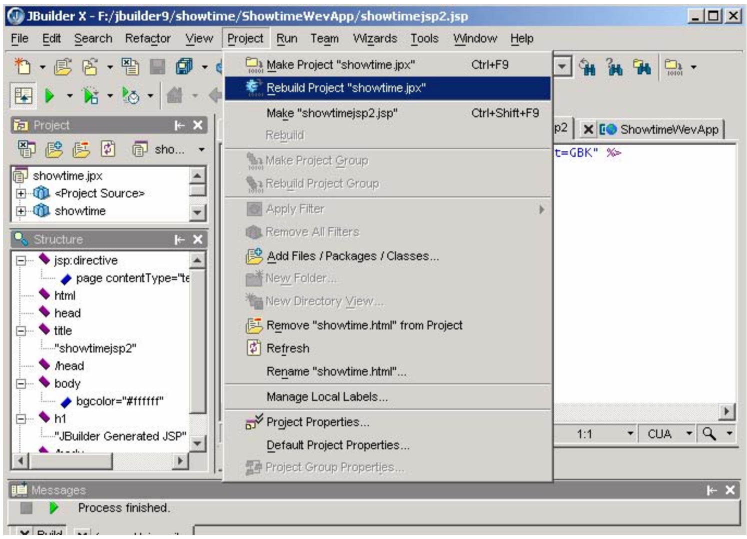


그림 6-17. JSP파일의 콤파일

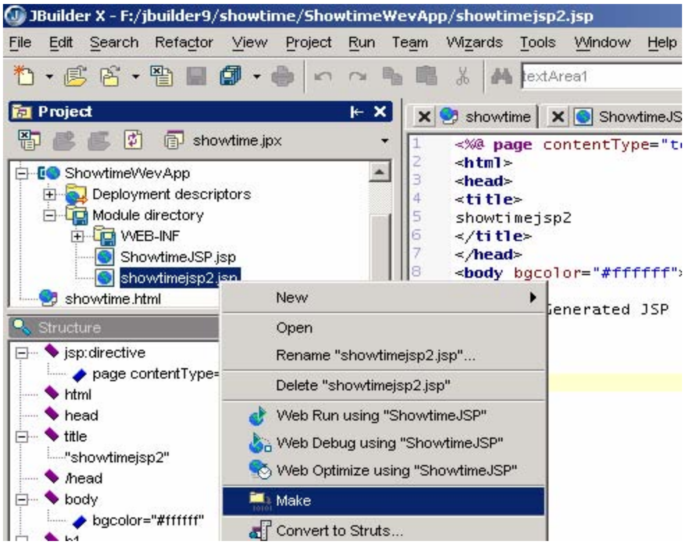


그림 6-18. 개별적인 JSP파일의 콤파일

6.4.2. JSP파일의 실행

프로젝트계층구조에서 실행하려는 JSP파일을 선택하고 마우스오른쪽단추를 찰칵하여 나온 지름차림표의 Web Run지령을 선택한다. 【Run】→【Run Project】지령을 리용하여 실행할수도 있다. 실행결과는 그림 6-19, 6-20과 같다.

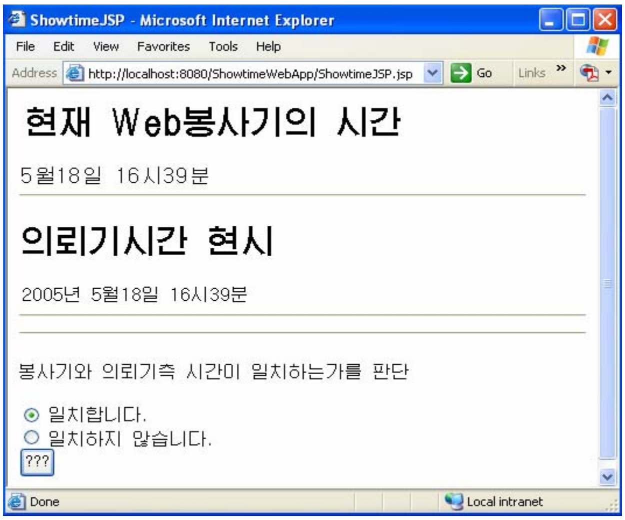


그림 6-19. 실행결과 1

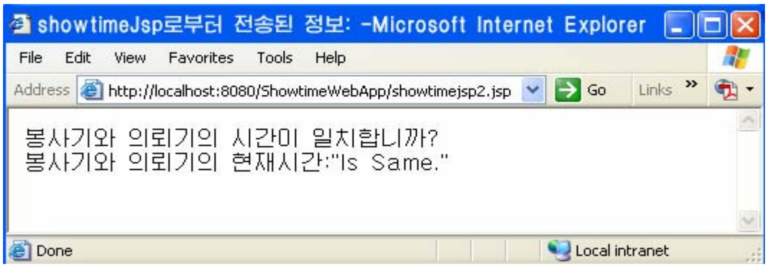


그림 6-20. 실행결과 2

6.4.3. 원천코드의 분석

아래에서 사용자들의 분석에 필요한 2개의 원천코드를 제시하였다.

- Showtimejsp.jsp원천코드

```
<%@ page contentType = “text/html;charset = Big5”%>
<%@ taglib unj = “http//java.sun.com/jstl/fmt”prefix = “fmt”%>
<html>
<head>
<title>
ShowtimeJSP
</title>
</head>
<jsp:useBean id = “ShowtimeJSPBeanid”scope = “session” class = “s
howtime.ShowtimeJSPBean”/>
<jsp: setProperty name = “ShowtimeJSPBeanid”property = “*”/>
<body bgcolor = “#ffffco”>
<h1>
현재 Web봉사기의 시간
</h1>
<%java.util.Date Mydate = new java.util.Date();%>
<%=Mydate.getMonth()+1%> 월
<%=Mydate.getDate()%> 일
<%=Mydate.getHours()%> 시
<%=Mydate.getMinutes()%> 분
<HR>
<H1>의뢰기시간 현시 </H1>
<Script language = ‘javaScript’>
Var clientDate = new Date() ;
```

```
document.write(clientDate.getYear()+ “년”);
document.write((clientDate.getMonth()+1)+“월”);
document.write(clientDate.getDate()+ “일”);
document.write(clientDate.getHours()+ “시”);
document.write(clientDate.getMinutes()+ “분”);
</script>
<HR>
<p>봉사기와 의뢰기의 시간이 일치하는가를 판단하시오</p>
<FORM ACTION= “showtimejsp2.jsp” METHOD= “POST”>
<INPUT TYPE= “radio” NAME= “showtimejsp” VALUE= “Is Same.”>
일치합니다.<BR>
<INPUT TYPE= “radio” NAME= “showtimejsp” VALUE= “Isn’t Sam
e.”>
일치하지 않습니다.<BR>
<INPUT TYPE= “submit” NAME= “submit” VALUE= “확인”>
</FORM>
</body>
</html>
```

- Showtimejsp2.jsp원천코드

```
<%@ Page ContentType = “text/html;Charset = Big5”%>
<html>
<head>
<title>
Showtimejsp.jsp에서 온 정보
</title>
</head>
<body bgcolor = “#ff0000”>
봉사기와 의뢰기시간이 일치합니까? <BR>
봉사기와 의뢰기의 현재시간 :
“<%=reguest.getParameter (“Showtimejsp”)%>”
<BR>
</body>
</html>
```

제5절. JSP의 배비

JSP의 배비는 비교적 쉽다. 그것은 웨브봉사기가 JSP를 찾는 방식이 HTML파일을 찾는 방식과 같기때문이다. 아래에서 배비과정을 간단히 소개한다.

단계

1【File】→【New】지령을 선택하면 Object Gallery대화칸이 현시된다. 여기서【Archive】항목을 선택하고 【Basic】아이콘을 선택한다.(그림 6-21)

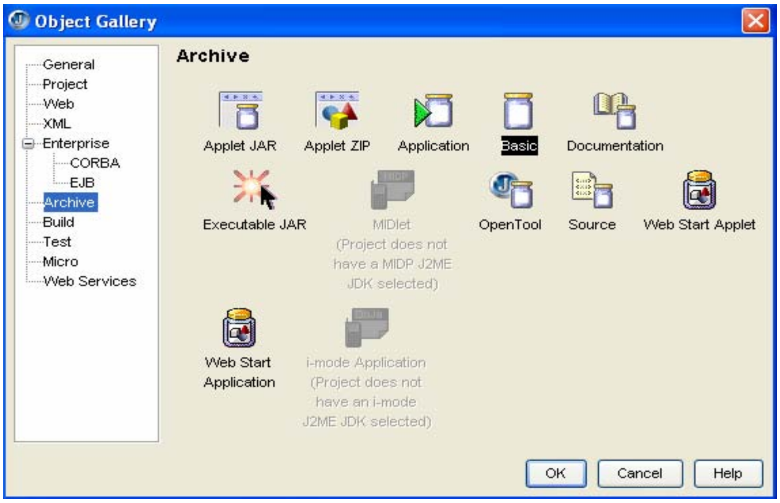


그림 6-21. Object Gallery대화칸의 Archive페지

2 【OK】단추를 찰칵하면 다음 페지로 넘어간다. 여기서 【Name:】본문칸에《showtimeArch》를 입력한다. 그 다음 경로를 지정해주고【Compress the contents of the archive】검사칸을 선택한다. 그리고 File Name의 보존파일이름을 showtime.jar로 한다. 다음 【Next】단추를 찰칵한다.(그림 6-22)

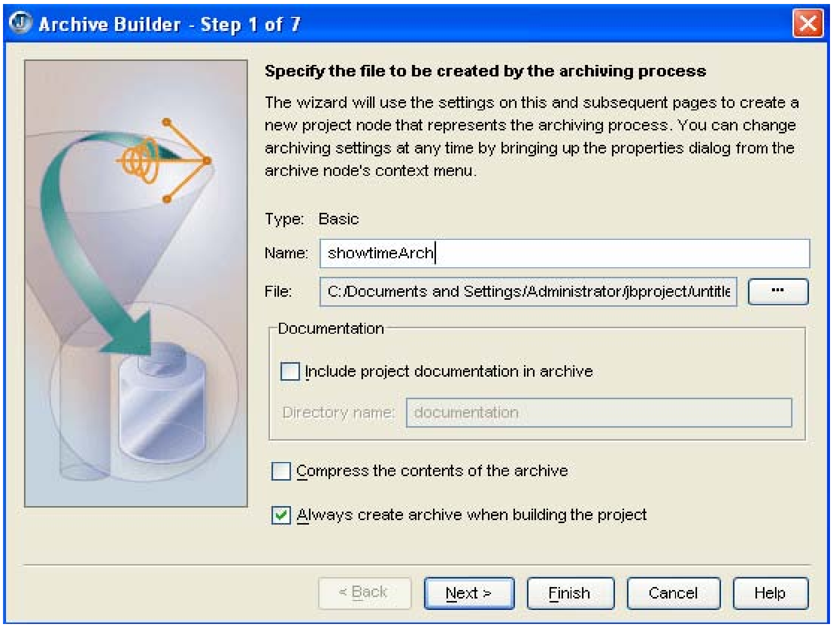


그림 6-22. Archive Builder대화칸의 첫 페지

3 기타 모든 추가선택항목들을 기정값으로 설정하고 【Finish】단추를 찰칵하여 Archive Buieler조수를 완성한다.

4 【Project】→【Make Project】지령을 선택하여 프로젝트를 콤파일한다. Archive Builder는 프로젝트출구등록부의 모든 파일들을 JAR파일로 만든다.

이렇게 하여 JSP의 배비를 완성한다.

제7장. Servlet응용

Servlet는 전통적인 CGI개발에서 또 하나의 비약을 가져왔다. Servlet의 매 요청은 하나의 새로운 프로쎄스에 의해서가 아니라 한개의 스레드에 의해 처리되는데 이것은 전통적인 CGI와 상반되는것이다. Servlet는 Java Servlet API를 리용하여 개발한다.

Java Servlet API는 javax.Servlet와 javax.Servlet.http패키지에 포함되여있다. 이 2개의 패키지에는 HTTP규약의 요청, 응답, 통신봉사 등의 기능을 수행할수 있는 모든 클라스, 대면, 기틀들이 포함되여있다. Java Servlet는 봉사기측프로그람작성에서 망통신처리와 관련한 패키지이다. Servlet가 Java로 된 프로그람이므로 봉사기프로그람개발자는 Java 및 그의 API가 가지고있는 모든 우점들을 리용할수 있다.

우점들은 다음과 같다.

- Servlet는 한번 작성하면 Java를 지원하는 임의의 봉사기에서 실행할수 있다.
- Java Database Connectivity(JDBC)를 리용하여 SQL자료기지를 관리할수 있다.

이 장에서는 JBuilder에 의한 Servlet프로그람작성방법들을 소개한다.

제1절. Servlet에 대한 개념

Servlet는 Java언어를 리용하여 실현한 규약과 가동기반에 의존하지 않는 봉사기용프로그람이다.

Servlet는 Java를 지원하는 봉사기를 동적으로 확장함으로써 Java를 지원하는 웨브봉사기에 여러가지 봉사기능을 보충해준다.

7.1.1. Servlet와 전통적인 CGI의 비교

Java Servlet는 전통적인 CGI기술보다 효률이 높고 사용하기가 편리하며 기능이 높고 가격이 눅은 우점들을 가지고있다.

- 효률측면: 전통적인 CGI는 매 HTTP에 대하여 새로운 프로쎄스(process)를 창조한다. 만일 어떤 CGI프로그람이 속도가 빠른 조작을 수행한다면 조작실행시간의 대부분은 프로쎄스를 창조하는데 소비된다. 하지만 Java Servlet는 매 요청을 하나의 프로쎄스가 아니라 하나의 Java스레드로 처리한다.

CGI는 같은 CGI프로그람에 대하여 N개의 요청을 받았을 때 같은 CGI프로그람을 기억기에 N번 불러들여야 한다. 그러나 Java Servlet는 이런 경우 N개의 스레드를 창조하

므로 해당한 Java Servlet를 클라스로서 한개만 기억기에 불러들이면 된다. 그 밖에 Java Servlet는 머리부계산, 자료기지련결 등에서 CGI보다 우월하다.

- 편리성측면: Java Servlet는 자동적으로 HTML자료의 추가 및 해석, HTTP머리부의 읽기와 설치, 쿠키처리, 대화접속추적 등 기타 많은 기능들을 가지고있다.
- 기능측면: Java Servlet는 CGI에서 실현할수 없거나 힘든 기능을 매우 쉽게 실현할수 있다. Servlet는 직접 봉사기와 통신을 진행할수 있지만 CGI는 그렇게 할수 없다.
- Java Servlet사이의 자료공유측면: Servlet사이에는 자료를 공유할수 있으며 자료기지에 접속하는것을 쉽게 실현할수 있다. 또한 Servlet는 요청과 응답에 이르기까지의 관리를 쉽게 실현할수 있다. 그러나 CGI는 매번 새로운 프로쎄스를 창조하고 파일을 리용하여 통신을 진행하여야 하므로 통신속도가 뜨다. 또한 하나의 봉사기에서 서로 다른 CGI프로그람사이의 통신 역시 매우 복잡하다.
- 호출시간측면: CGI프로그람은 독자적인 프로쎄스에 의하여 실행되므로 호출시간이 길다. 그러나 기억기속에 들어있는 Servlet는 불러들이는 시간이 매우 짧다.
- 보안측면: 일부 CGI프로그람은 보안상 약점을 가지고있다. perl과 같은 언어를 리용한다고 해도 체계는 기본적인 보안시설을 가지고 있지 못하고 어떤 규칙에 의거할뿐이다. 그러나 Java는 SSL, CA인증, 보안정책, 규범 등 완전한 보안체계를 가지고있다.
- 기교측면: Java Servlet프로그람은 순수한 Java로 작성하였다. 그러므로 거의 모든 봉사기들에서 직접적 혹은 보조적으로 실행될수 있다.
- 원가측면: 많은것들이 공개이므로 원가가 눅다.

7.1.2. Servlet에 의해 새롭게 추가된 봉사특징

새로 보충된 봉사특징은 다음과 같다.

- 홈페지내용의 동적변화(Runtime Changes)
- 홈페지현시의 동적변화와 새로운 표준규약에 대한 지원(례를 들어 FTP)
- 사용자의 규약지원 보증

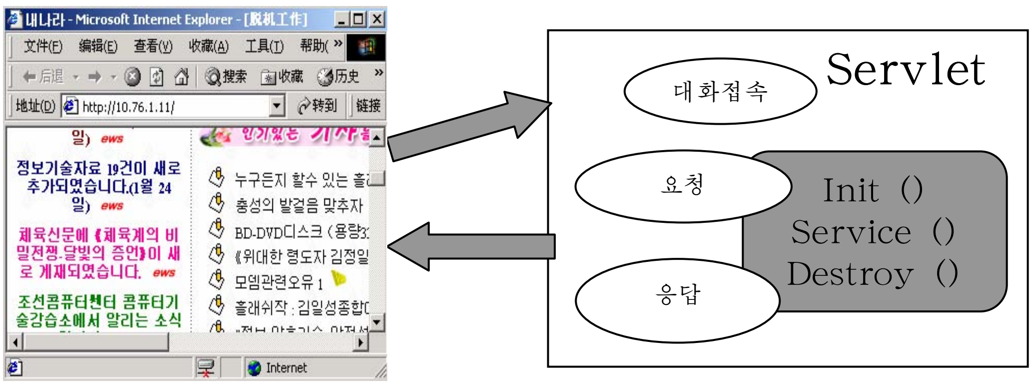


그림7-1. Servlet실행도

그림 7-1에 보여준바와 같이 Servlet는 요청/응답(Request/Response)방식으로 작업한다. 이 방식에서 의뢰기는 봉사기에 요청정보를 보내고 봉사기는 이에 대한 응답정보를 발송한다.

Servlet가 의뢰기로부터 요청을 받을 때 이것은 ServletRequest와 ServletResponse를 받아 의뢰기로부터의 요청과 응답을 처리해준다. ServletRequest클라스는 의뢰기에서 봉사기에로의 련결을 보장해주고 ServletResponse는 봉사기에서 의뢰기에로의 련결을 보장한다.

ServletRequest대면은 의뢰기가 보내는 의뢰기이름, 그것이 사용하는 규약, 요청, 원격봉사기의 이름 등의 정보를 받는다. 이것은 또한 Servlet, Servlet Stream을 제공하는데 의뢰기측에서는 POST, PUT메쏘드들로 이 자료흐름을 만든다. Servlet는 ServletRequest의 하위클라스를 리용하여 여러가지 규약의 특성자료들을 많이 얻을수 있다. 실례로 HttpServletRequest는 HTTP-specific머리부정보를 얻는 메쏘드를 가지고있다.

ServletResponse대면은 봉사기측의 Servlet메쏘드들을 가지고있다. 이 메쏘드들은 Servlet가 내용의 길이, 응답의 mime류형을 설정하게 하며 출력흐름인 Servlet Output Stream을 제공한다. Servlet Response하위클라스는 많은 protocol-specific정보를 얻을수 있게 한다.

HTTP Servlets는 Session-tracking capabilities메쏘드를 제공하는데 Servlet프로그람작성자는 이러한 API를 리용하여 Servlet와 의뢰기사이의 상태를 관리할수 있다.

요청정보는 아래와 같은 규약을 리용하여 전달된다.

- HTTP
- URL
- FTP
- 사용자계종규약

일반적으로 요청과 그에 대한 응답은 요청시 의뢰기와 봉사기의 상태를 반영한다. Servlet는 여러개의 요청/응답의 대화정보를 보관할수 있다.(그림 7-2)

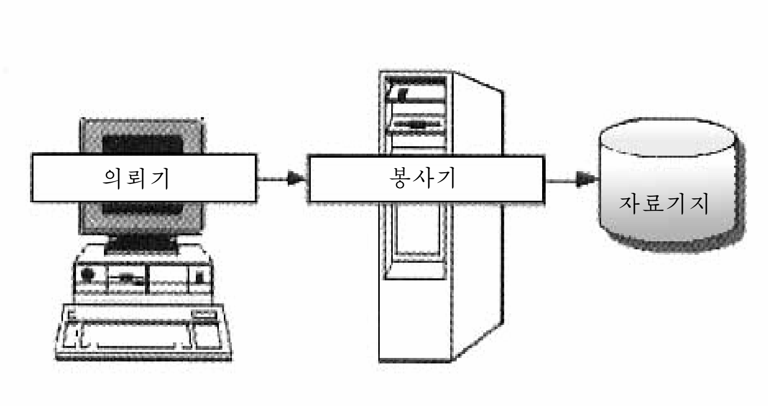


그림 7-2. Servlet의 간단한 응용

Java Servlet API는 기본 봉사기와 Servlet사이의 련결을 정의해주는 몇개의 Java 대면(Java Interfaces)들을 포함하고있다. Servlet API는 표준 Java개발패키지(JDK)에 속한다. 이것은 javax.Servlet와 javax.Servlet.http에 포함되여있다.

Servlet API는 대화접속추적(Session tracking), 공유자료(Shared Data), 국제표준화부호 I/O, 파라메터초기화, 요청대리(Reguest Delegation) 등 필요한 기능들을 많이 포함하고있다.

7.1.3. Servlet의 생명주기

Servlet는 소응용프로그람과 같이 생명주기를 가지고있다. Servlet의 생명주기는 봉사기가 Servlet를 실행하는 동안의 시간을 말한다. Servlet는 의뢰기의 요청들을 접수하고 그에 응답한 다음 끝낸다. 아래에서 이 내용에 대하여 자세히 설명한다.

- 초기화

봉사기가 Servlet를 적재할 때 Servlet의 init()메쏘드를 호출한다.

```
public void init(ServletConfig config) throws SerlvetException
{
super.init( ); // 일련의 초기화조작
}
```

끝마칠 때 super.init()를 호출하여야 한다. init( )메쏘드는 반복하여 리용할수 없다. 일단 이 메쏘드를 호출하면 Servlet를 재적재한다.

- Servlet의 실행

Servlet를 초기화한 다음 Servlet는 의뢰기의 요청을 처리한다. 이 조작은 Service 메쏘드를 리용하여 진행한다. 매 의뢰기는 자기의 봉사메쏘드들을 리용하여 봉사기에 요청을 보낸다. Servlet는 동시에 여러개의 봉사를 실행할수 있다. 봉사메쏘드는 thread-safe 형식으로 작성할수 있다.

만일 어떤 봉사기가 Service메쏘드를 동시에 호출할수 없으면 SingleThreadModel 대면을 리용할수 있다. 이 대면은 2개이상의 스레드가 병렬로 실행되는것을 막는다. Servlet의 작성에서는 의뢰기의 요청과 망페지의 생성이 가장 많은 자리를 차지한다.

코드는 아래와 같다.

```
printWriter out = response.getWriter();
out.println("");
out.println("");
out.println("Hello World");
out.println("");
out.close();
```

- Servlet의 끝내기

Servlet는 destroy( )메쏘드를 리용하여 끝내는데 이때 init( )메쏘드에서 리용한 자원들을 회수한다.

이 메쏘드의 형식은 아래와 같다.

```
public void destroy()
{
//자료기지 련결, 차단과 같은 init()에서 초기화한 자원을 회수한다.
}
```

7.1.4. Servlet의 특징과 응용

Servlet는 여러가지 우점들을 가지고있다. Servlet는 현재의 봉사기에서 실행되기때문에 이미 작성된 코드와 기술을 효과적으로 리용하며 망련결, 규약통합, 클라스적재 등의 작업을 봉사기가 진행하게 하여 반복작업을 감소시킨다.

1. Servlet의 다양한 응용

Servlet는 여러개의 요청을 병렬로 처리할수 있다. 복잡한 작업은 여러개의 활동대행체(active agents)로 처리할수 있는데 매 대행체는 Servlet를 대신하여 실행적재되며 매 대행체사이에 통신도 진행할수 있다.

Servlet는 요청을 기타 다른 봉사기 혹은 Servlet에 넘겨줄수 있다. 이 기술은 같은 봉사를 진행하는 여러개의 봉사기들사이에 부하균형을 이룰수 있게 한다. Servlet의 이러한 특성으로 하여 중간층(middle-tier)에 기초한 응용처리와 대리(Proxy)처리를 할수 있으며 그리고 새로운 규약과 특정한 봉사를 리용하여 중간층의 봉사를 개선할수도 있다.

2. 중간층처리

일반적으로 말하는 3층 의뢰기/봉사기체계(three-tier client/server systems)에서 중간층은 응용봉사기(Application Server)로서 앞단의 의뢰기(례를 들어 웨브열람기)와 뒤단의 봉사 및 자료를 련결해준다. 중간층을 리용하여 앞뒤단을 갈라놓으므로써 앞단의 의뢰기가 가볍고 속도가 빠르게 하며 뒤단의 봉사는 자기의 과제를 정확히 집중적으로 수행할수 있게 한다. Servlet는 중간층처리를 진행하는데 매우 적합하다. Servlet의 우점은 련결관리를 간략화하고 자료기지관리를 효과적으로 진행하여 수천수백의 의뢰기련결을 쉽게 처리할수 있다는것이다.

— 중간층에서 Servlet의 기타 역할

- 업무규칙의 강화(Business rule enforcement)
- 사무관리(Transaction Management)
- 의뢰기와 봉사기사이의 넘기기(mapping)

- 서로 다른 류형의 의뢰기들을 지원(례; 순수한 HTML과 Java권한사용 의뢰기)

3. 대리봉사기

Servlet는 Applets를 대리할수 있다. Java보안기구는 Applets들이 현지에서 그것들의 내리적재봉사기까지의 접속에서만 보안을 허용하므로 이 특성은 매우 중요하다. 만일 하나의 Applet가 다른 호스트콤퓨터에 위치한 자료기지봉사기와의 접속을 시도하면 Servlet는 그것을 대신하여 우의 작업을 완성할수 있다.

4. 규약지원

Servlet API는 봉사기와 Servlet사이의 긴밀한 련계를 보장하기 위하여 봉사기에 새로 추가된 규약을 지원해준다. Servlet API패키지는 이미 HTTP를 지원하고있다. 다시 말하여 SMTP, POP, FTP와 같은 요청/응답형식의 규약은 Servlet를 리용하여 실현할수 있다.

현재 기본적인 웨브봉사기들은 Servlet를 지원하며 점점 더 많은 류형의 응용프로그람봉사기들이 Servlet를 지원하고있다. HTTP가 가장 많이 쓰이는 규약인것으로 하여 Servlet는 HTTP체계에서 가장 광범히 응용되고있다.

7.1.5. Servlet의 대면

Servlet는 javax패키지의 httpServlet클라스의 하위클라스이다. HttpServlet는 다음의 메쏘드들을 가지고있다.

- doGet

이 메쏘드는 HTTP GET요청을 처리한다.

doGet는 다음과 같이 정의한다.

```
protected void doGet(HttpServletRequest reqest,
     HttpServletRespone response) throws ServletException,
     IOException;
```

- doPost

이 메쏘드는 HTTP의 POST요청을 처리한다.

Servlet를 작성할 때 POST조작을 지원하려면 HttpServlet의 하위클라스에서 이 메쏘드를 리용하여야 한다.

doPost메쏘드는 다음과 같이 정의한다.

```
protected void doPost(HttpServletRequest request
     HttpServletResponse response)throws ServletException;
     IOException;
```

- doPut

이 메쏘드는 HTTP의 PUT요청을 처리한다. 이 PUT조작은 FTP를 통하여 파일을 보내는데 쓰인다.

메쏘드는 다음과 같이 정의한다.

```
protected void doPut(HttpServletRequet request,
      HttpServletResponse response) throws ServletException;
      IOException;
```

- doDelete

이 메쏘드는 HTTP의 DELETE요청을 처리한다.

```
protected void doDelet(HttpServletRequest request,
      HttpServletResponse response)throws ServletException,
      IOException;
```

- doHead

이 메쏘드는 HTTP의 HEAD요청을 처리한다.

```
protected void doHeat(HttpServletRequest request,
      HttpServletResponse response)throws ServletException,
      IOException;
```

- doOptions

이 메쏘드는 HTTP의 OPTIONS요청을 처리한다.

```
protected void doOptions(HttpServletRequest request,
      HttpServletResponse response) throws ServletException,
      IOException;
```

- doTrace

이 메쏘드는 HTTP의 TRACE요청을 처리한다.

```
protected void doTrace(HttpServletRequest request,
      HttpServletResponse response)throws ServletException,
      IOException;
```

HTTP를 기본으로 하는 Servlet를 개발하는 경우 doGet, doPost메쏘드들이 많이 쓰인다.

7.1.6. HTTP대화접속

웨브봉사기응용프로그람은 원격의뢰기의 요청을 식별하고 처리할수 있도록 작성해야 한다.

Java Servlet API는 간단한 대면을 제공함으로써 Servlet에서 메쏘드들을 사용하여 사용자의 대화접속을 감시할수 있게 하였다

1. 대화접속창조

HTTP는 요청/응답형식의 규약으로서 의뢰기의 가입을 새로운 대화접속으로 본다. 여기에서 가입(join)은 의뢰기의 대화접속감시기정보가 봉사기에 전달되는것을 의미하며 이때 대화접속이 이루어졌다고 한다.

Servlet개발자는 반드시 의뢰기가 대화접속에 가입하지 않았거나 가입할수 없는 경우를 처리할수 있도록 웨브응용프로그람을 설계하여야 한다. 봉사기는 일정한 대화접속시간을 정하여 그 시간이 지나면 대화접속을 중지시킬수 있다.

2. 객체를 대화접속에 대응시키기

객체를 대화접속에 련관시켜 놓을 필요가 제기될수 있다. HttpSession객체를 리용하여 임의의 객체를 대화접속에 련관시켜 놓으면 이 객체는 동일한 대화접속으로부터 오는 임의의 요청 Servlet에 대하여서만 유효하다. 어떤 객체들은 대화접속의 창조와 삭제시간을 요구할수 있는데 이때 HttpSessionBindingListener대면을 리용하여 이 정보를 얻는다.

3. Servlet에서 대화접속사건얻기

대화접속사건얻기는 홈페지의 사용자등록일지(등록, 탈퇴 등의 정보), 현재 등록자수를 통계내는데 사용된다. 대화접속은 의뢰기의 대화접속과정을 대표한다. 의뢰기등록시 Session에 하나의 객체를 대입시켜 대화를 추적한다. 만일 HttpSessionBindingListener 대면(편리를 위하여 감시기라고 부른다.)을 실현하면 servlet에 대화접속을 창조할 때 (HttpSession객체의 setAttribute메쏘드를 호출한 경우)와 대화접속을 삭제할 때 (HttpSession객체의 removeAttribute메쏘드를 호출하거나 시간이 초과된 경우) Session 객체는 자동적으로 감시기의 valueBound와 valueUnBound메쏘드(HttpSession BindingListener의 메쏘드)를 호출한다. 이것을 리용하면 등록일지를 쉽게 실현할수 있다.

다른 하나의 문제는 망에 련결된 사용자수를 계수하는것이다. 이것은 사용자등록일지를 실현하는것과 조금 차이난다. 사용자수는 존재하는 Session수와 같다. 아래의 실례에서는 valueBound와 valueUnBound메쏘드를 리용하여 대화접속창조와 삭제시 계수기에 1을 더해주거나 덜어주는 방법으로 사용자수를 계수한다.

다음의 코드는 Session을 리용한 감시기실례이다.

```
//SessionListener.java
  import java.io.*;
```

```
import java.util.*;
  // 등록과정감시
  public class SessionListener implements HttpSessionBindingListener
   {
     public String privateInfo= “ ” ; //감시기의 초기화파라메터
     private String longString= “ ” ; //기록문자렬
     private int count=0;//등록계수기
     public SessionListener(String info){
         this.privateInfo=into;
     }

   public int getCount(){
     return count;
   }
   public void valueBound(HttpSessionBindingEvent  event)
   {
   count++;
   if(privateInfo.equals(«count»))
   {
      return;
    }
    try{
    Calendar calendar= new GregorianCalendar();
    System.out.println( “LOGON:” +privateInfo+
                         + “TIME:” +calendar.getTime());
     logString= “\nLOGON:” +privateInfo+ “TIME:” +
                         +calendar.getTime()+” \n” ;
    for(int i=1;i<1000;i++){
      File file =new File( “logfile.log” +i);
    if(!(file.exists())
      File.createNewFile();//파일이 존재하지 않으면 창조한다.
    if(!file.length()>1048576)//파일의 크기가 1M이상이면
                                 새로운 파일을 창조
      continue;
      FileOutputStream foo=new FileOutputStream
            ( “logfile.log” +i,true);  //추가방식으로 파일을 창조
```

```
 break;
 }
}
catch(FileNotFoundException e{}
catch(IOException e{}
}
public void valueUnbound(HttpSessionBindingEvent  event)
{
count--;
if(privateInfo.equals( “count” ))
{
return;
}
try{
Calendar calendar = new GregorianCalendar();
System.out.println( “LOGOUT:” +privateInfo+
                    + “TIME:” +calendar.getTime());
logString= “\nLOGOUT:” +privateInfo+ “TIME:” +
                    +calendar.getTime()+ “\n” ;
for(int i=1;i<1000;i++){
File file=new File( “logfile.log” +i);
if(!(file.exists()))
File.createNewFile();
if(file.length()>1048576)
continue;
FileOutputStream foo=new FileOutputStream
                      ( “logfile.log” +i,true);
foo.close();
break;
}
}
catch(FileNotFoundException e){}
catch(IOException e){}
}
}
```

Servlet에서 이 감시기를 리용하는 코드는 다음과 같다.

```
    .............
    HttpSession session=req.getSession(true);
    .............
    ///////////////////////////////////////
    SessionListener sessionListener=
                 new SessionListener( "IP:" +req.getRemoteAddr();
    session.setAttribute( "listener" ,sessionListener);
   ////////////////////////////////////////////
```

체계가 등록에서 탈퇴할 때 session.removeAttribute("listener")를 호출하면 감시기의 valueUnbound메쏘드가 자동적으로 호출된다. 대화접속의 존재기한이 지났을 때(Session Time Out)에도 이 메쏘드가 호출될수 있다.

매 대화접속가입자수를 통계내는 코드는 다음과 같다.

```
ServletContext session1=getServletConfig().getServletContext();
//ServletContext객체얻기
if((SessionListener)session1.getAttribute( "listener1" )==null
{
SessionListener sessionListener1=new SessionListener( "count" );
//의뢰기가 봉사기와 련결할 때 한번만 설정해준다.
session1.setAttribute( "listener1" ,sessionListener1);
//감시기객체를 ServletContext의 속성으로 설정해준다.
}
session.setAttribute( "listener1" ,(SessionListener)session1.getAttribute
( "listener1" ));
```

아래의 코드를 리용하여 프로그람의 임의의 장소에서 당시 등록자수를 얻을수 있다.

((SessionListener)session.getAttribute("listener1")).getCount();

제2절. Servlet프로그람작성

일반적으로 Servlet는 HTTP규약을 리용하여 웨브봉사를 실현한다. 이때 사용되는 클라스가 javax.Servlet.http.HttpServlet이다.

HttpServlet클라스는 GenericServlet클라스를 계승하며 Servlet대면을 실현한다. 이것은 HTTP/1.1의 요청을 지원한다.

HttpServlet클라스를 리용하여 작성한 Servlet는 동시에 여러개의 스레드를 리용하여 봉사를 진행할수 있다. 만일 하나의 스레드로 하나의 봉사프로그람을 실현하려면 HttpServlet를 계승할뿐아니라 SingleThreadMode대면을 실현하여야 한다.

실례로

```
public class surveyservlet  extends HttpServlet
     implements SingleThreadMode
{
/*전형적인 servlet코드들은 service메쏘드에서 서술된다.
}
```

7.2.1. 의뢰기와의 호상작용

Httpservlet클라스를 리용하여 Servlet프로그람을 작성하는 경우 HTTP메쏘드들을 리용하여야 한다.

그 메쏘드들에는 다음과 같은것들이 있다.

- doGet: GET요청과 HEAD요청을 처리한다
- doPost: POST요청을 처리한다
- doDelete: DELETE요청을 처리한다

기정값일 때 이 메쏘드들은 BAD-REQUEST(400)오유를 귀환한다.

또한 HttpServlet의 Service메쏘드는 doOption메쏘드(OPTIONS요청을 받을 때)와 doTrace메쏘드(TRACE요청을 받을 때)를 호출한다. doOption은 자동적으로 지원하여야 할 HTTP항목을 결정하고 이 정보를 되돌려준다.이 두 메쏘드들은 2개의 파라메터를 요구하는데 첫 파라메터는 의뢰기에서 보낸 자료로서 HttpServletRequest이다.

HttpServletRequest는 HTTP제목정보에 대한 관리를 진행한다.

의뢰기자료관리는 HTTP메쏘드에 의해 결정된다.

- 요청에 대하여서는 getParameter(Values)메쏘드를 리용하여 파라메터를 얻을수 있다.

• HTTP GET메쏘드를 사용하는 경우 요청에 대하여 getQueryString을 리용하여 파라메터를 얻을수 있다.

• HTTP의 POST, PUT, DELETE에 대해서는 getReader와 getInputStream메쏘드 중 어느 하나를 리용할수 있다. 만일 본문자료를 원한다면 getReader메쏘드가 귀환하는 bufferReader를 리용하여 얻을수 있다. 한편 2진자료를 원한다면 getInputStream이 귀환하는 ServletInputStream을 리용하여 얻을수 있다.

의뢰기의 응답은 HttpServletResponse객체를 리용한다. 이때 getWriter 혹은 getOutputStream을 리용하여 내용을 볼수 있다. HttpServletResponse클라스는 제목정보와 응답내용의 길이를 제공해준다.

7.2.2. 생명주기메쏘드

Servlet프로그람은 init, service, destroy인 3가지 메쏘드에 의하여 초기화, 봉사, 파괴의 과정을 거친다는것을 우에서 언급하였다. 그림 7-3에서 보는바와 같이 Servlet가 적재될 때 init메쏘드를 한번만 호출하며 의뢰기의 요청이 있을 때마다 service메쏘드를 반복적으로 호출하게 된다. 이 3가지 메쏘드는 Servlet대면의 가장 기본적인 메쏘드이며 Servlet의 생명주기와 관련된 메쏘드들이다.

Servlet에 요청이 처음으로 들어왔을 때 해당 Servlet는 Servlet용기에 의해 자동으로 기억기에 적재된다. 기억기에 Servlet클라스가 적재된 다음 객체를 생성하며 객체의 생성과 동시에 init메쏘드를 호출한다. init메쏘드는 Servlet적재시 한번만 호출되며 오유가 발생했을 때 UnavailableException례외나 ServletException례외를 발생시킨다.

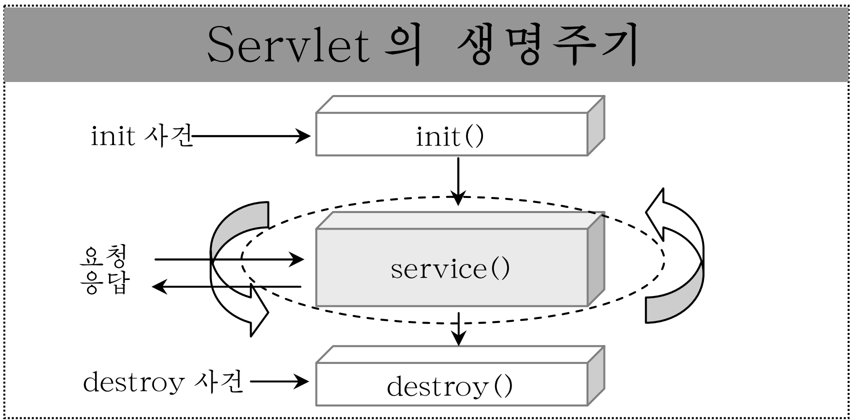


그림 7-3. Servlet의 생명주기

Servlet객체는 기억기를 해제하는 경우 destroy메쏘드를 호출한다.

만약 성공적으로 init메쏘드가 호출되였다면 service메쏘드를 수행하여 의뢰기의 요청에 반응한다. 두번째 의뢰기의 요청이 있을 때부터는 service메쏘드를 호출하여 의뢰기의 요

청에 응답한다. service메쏘드는 의뢰기의 요청방식에 따라 Get방식이면 doGet메쏘드를, Post방식이면 doPost메쏘드를 호출한다. Servlet객체가 더 이상 봉사를 진행할 필요가 없는 경우 기억기에서 제거하기 위하여 destroy()메쏘드를 호출한다. destroy()메쏘드가 호출되면 페품수집기는 객체의 기억기를 제거한다.

제3절. Servlet의 구조

아래에서 Servlet의 기본구조를 간단히 보여주는 코드를 주었다. 이 Servlet는 GET 요청을 처리한다.

```
import java.io*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SomeServlet extends HttpServlet{
public void doGet(HttpServletRequest request,HttpServletResponse
          response) throws ServletException,IOException{
// "request"를 리용하여 필요한 정보(실례로 Cookies) 와 표자료를 읽어들이거나 처
리를 진행한다.
// "response"를 리용하여 HTTP응답상태코드와 응답머리부를 지정해준다.
PrintWriter out= response.getWriter();
// "out"를 리용하여 응답내용을 열람기에 발송한다
}
```

만일 어떤 클라스가 Servlet로 되자면 HttpServlet를 계승하여야 하며 doGet, doPost 메쏘드를 재정의하고 GET, POST방식으로 자료를 발송한다.

doGet, doPost메쏘드는 각각 2개의 파라메터 즉 HttpServletRequest와 HttpServletResponse클라스형의 파라메터를 가진다.

HttpServletRequest는 요청정보를 제공하며 HttpServletResponse는 HTTP응답상태(200, 400 등)와 응답머리부(Content-Type, set-Cookie 등)를 정해주는 외에 의뢰기에 자료를 발송하는데 리용하는 PrintWriter를 제공해준다.

doGet와 doPost메쏘드들은 모두 service메쏘드에 의해 호출된다.

Servlet가 GET와 POST인 2가지 요청을 처리해야 하는 경우 Service메쏘드를 재정의할수 있다. Servlet는 Servlet API를 리용하여 창조한 특수한 부품이며

javax.Servlet.GenericServlet 혹은 javax.Servlet.http.HttpServlet를 확장한것이다.

javax.Servlet.GenericServlet는 규약에 의존하지 않는 보통 Servlet를 정의한 클라스이며 javax.Servlet.http.HttpServlet는 이 클라스의 하위클라스로서 HTTP의 메쏘드를 제공한다. 그림 7-4는 HttpServlet의 전형적인 계층구조와 매 층의 기능, 그리고 클라스들사이의 관계를 보여준다.

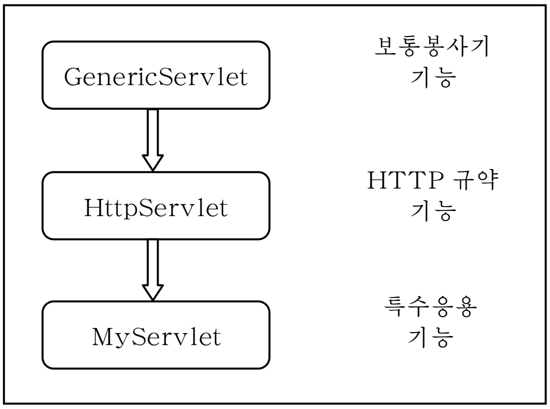


그림 7-4. 클라스의 계층구조

그림에서 보여준바와 같이 GenericServlet클라스는 봉사기의 일반조작을 정의해주었고 HttpServlet는 이 클라스를 계승하여 HTTP의 전용적인 조작을 제공한다.

아래에서 의뢰기와 Servlet사이의 조작에 대하여 소개한다.(그림 7-5)

전형적인 HTTPServlet환경에서 사용자가 웨브열람기에서 마우스를 누르면 봉사기측의 Servlet를 기동하여 URL로서 Servlet와의 대화접속을 시작한다. 열람기는 소케트의 80포구를 통하여 Servlet에 요청한다. Servlet는 80포구를 감시하다가 요청에 응답한다. 이때 요청에는 필요한 파라메터와 자료들이 포함되며 응용프로그람흐름을 사건처리부분에 보내여 하나 혹은 여러개의 과제를 수행한다. 그 결과를 사용자의 열람기에 보낸다.

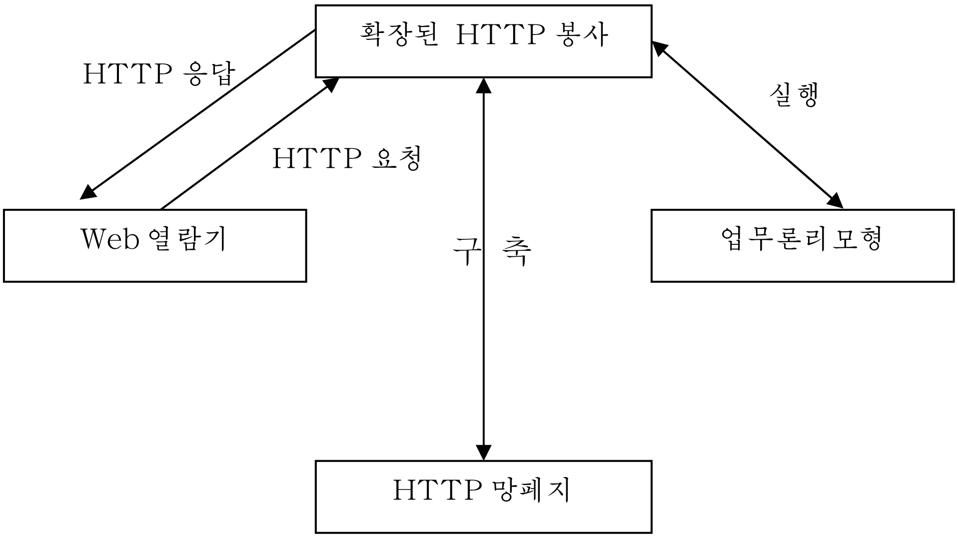


그림 7-5. Servlet조작흐름

제4절. Servlet조수를 리용한 Servlet의 작성

이 절에서는 Servlet를 어떻게 창조하는가에 대하여 설명한다. JBuilder에서는 웨브봉사기를 리용하지 않고도 Servlet를 개발하고 실행할수 있다. Servlet를 설치하려면 Tomcat와 같은 Servlet를 지원하는 웨브봉사기를 기동하여야 한다.

Java Servlet개발방법을 HelloWorld응용프로그람(firstServlet)을 실례로 설명하기로 한다. firstServlet는 환영통보문, 사용자이름, Servlet가 기동한 후 사용자와의 련결부를 현시한다.

모든 Servlet는 기본 Servlet클라스를 계승하며 입력련결처리에 Java메쏘드를 작성하여 리용한다. 실례로 Servlet클라스는 HttpServlet클라스를 계승하고있는데 이 클라스는 웨브의 HTTP규약에 맞게 웨브응용프로그람에 필요한 거의 모든 기능들을 처리할수 있다.

JBuilder에서 프로그람작성순서는 아래와 같다.

단계

1 【File】→【New Project】를 선택하고 【Name:】본문칸에 《myfirstservlet》를 입력한 다음 【Finish】단추를 찰칵한다.

2 【File】→【New】지령을 선택하면 【Object Gallery】대화칸이 나타난다. 여기서 Web항목을 선택한 다음 오른쪽 부분에 있는 Standard Servlet아이콘을 두번 찰칵한다.(그림 7-6)

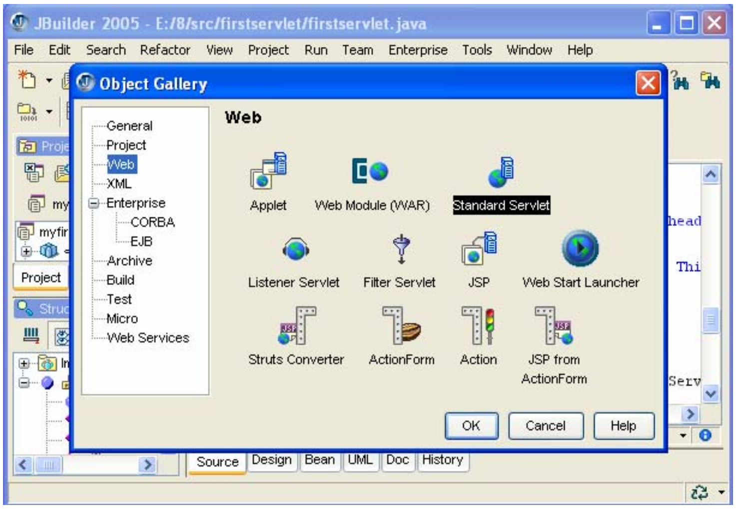


그림 7-6. Object Gallery대화칸의 Web페지

3 이때 봉사기설정을 위한 대화칸이 나타난다.(그림 7-7) 여기서 Tomcat4.1봉사기를 선택한 다음 【OK】단추를 찰칵하여 Servlet Wizard대화칸을 연다.(그림 7-8)

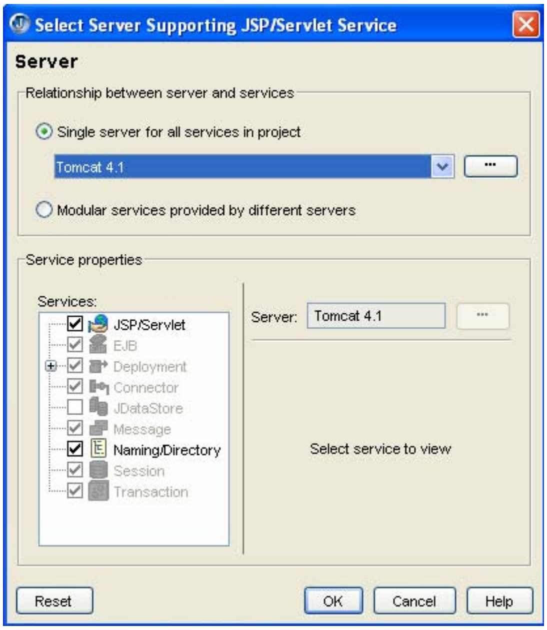


그림 7-7. 봉사기설정대화칸

4 이 페지에서 패키지이름 firstServlet와 클라스이름 Servlet1을 입력한다. 그리고 【Web module】복합칸이 비였을 때에는 【New】단추를 찰각하고 기정으로 추가선택항목들을 선택하여 Web모듈을 추가하여야 한다.

5 2번째 페지에서는 【Servlet create content type】복합칸에서 《HTML》를 선택한다. 또한 Servlet메쏘드로 doGet(), doPost()메쏘드들을 선택한다.(그림 7-9)

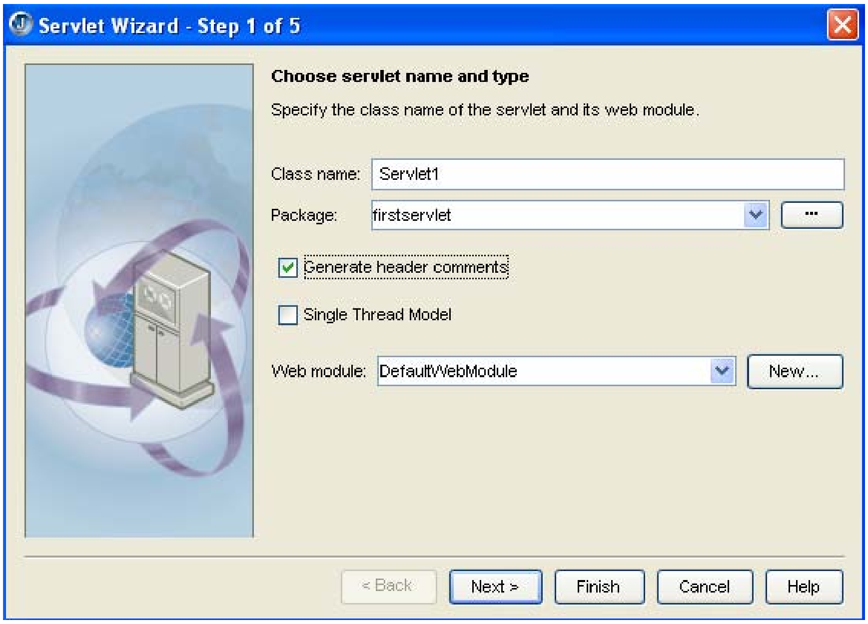


그림 7-8. Servlet Wizard대화칸의 첫 페지

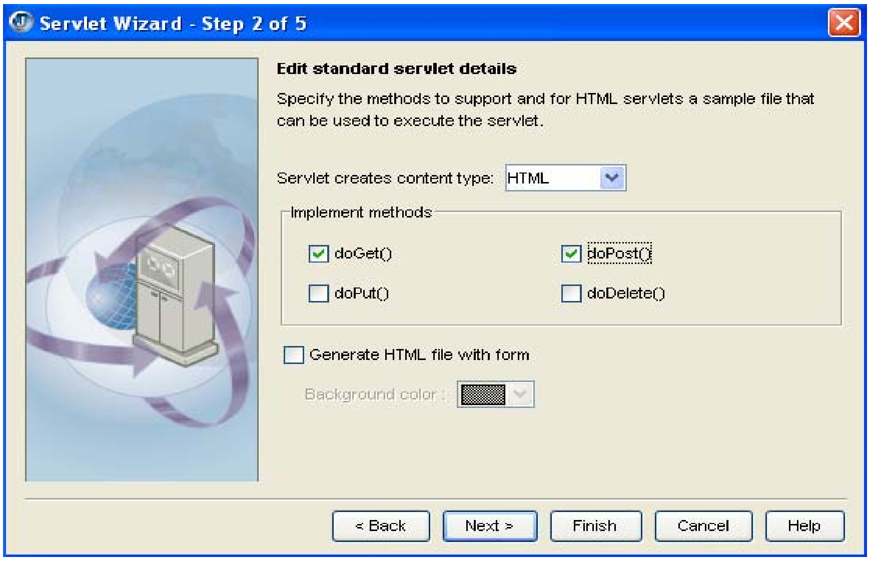


그림 7-9. Servlet Wizard대화칸의 2번째 페지

6 3번째 페지에서 표 7-1과 같이 내용을 입력한다.(그림 7-10)

표 7-1. Servlet 파라메터

Name	Param0
Type	String
Desc	NameofUser
Variable	Username
Default	Hello, kim

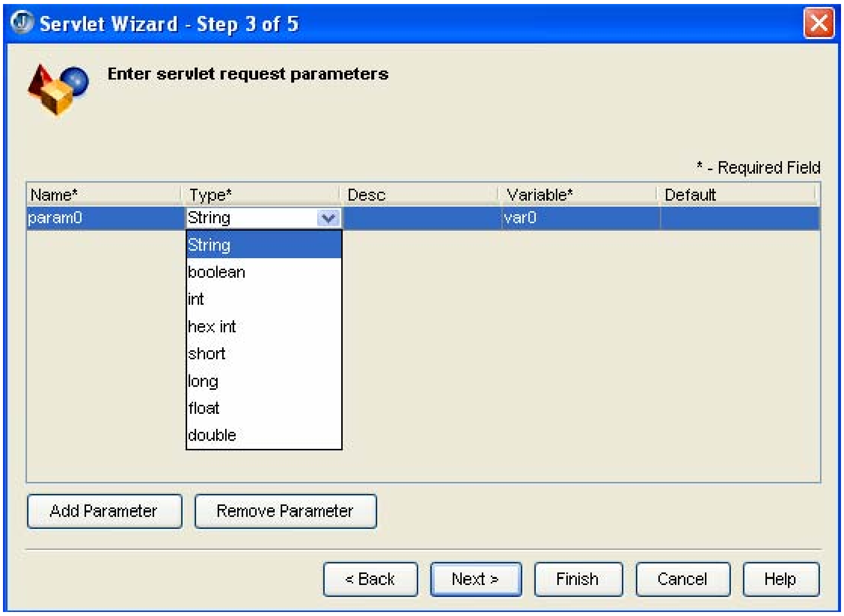


그림 7-10. Servlet Wizard대화칸의 3번째 페지

7【Next】단추를 찰칵하면 Servlet Wizard의 4번째 페지에로 넘어간다.(그림 7-11)

8 추가선택항목들을 기정으로 선택하고 【Finish】단추를 찰칵하면 Servlet편집상태에 들어간다.(그림 7-12)

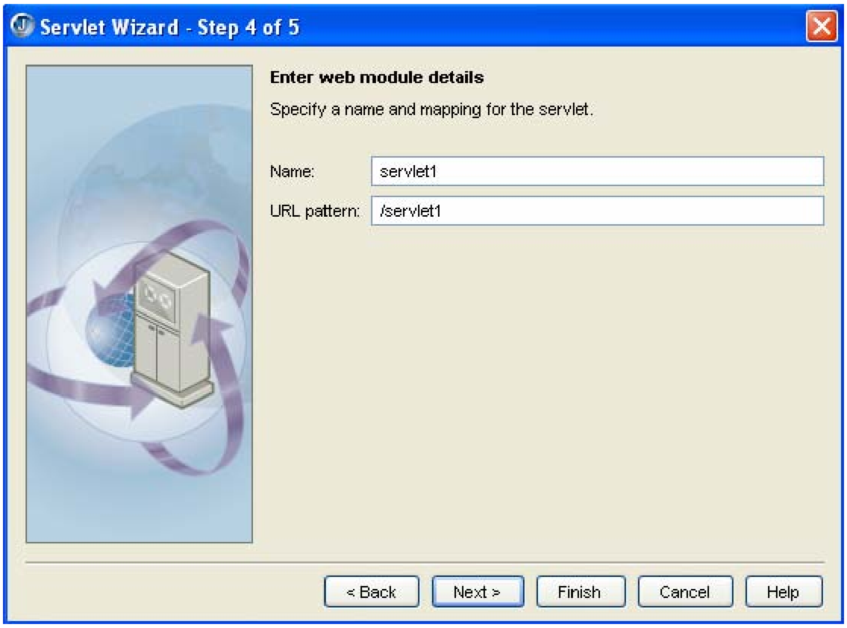


그림 7-11 Servlet Wizard대화칸의 4번째 페지

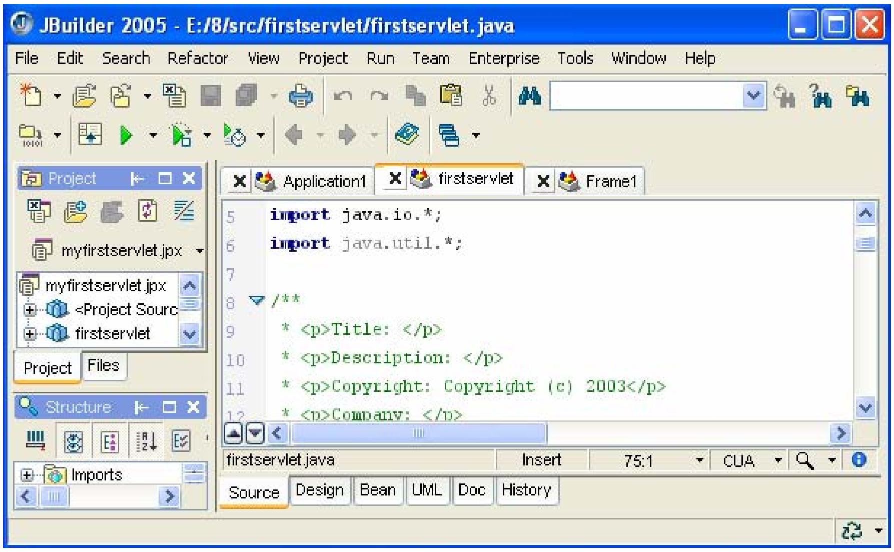


그림 7-12. Servlet편집창

이때 이름이 firstServlet.java, ServletServer.java, firstServlet.html인 파일들이 프로젝트에 추가된다.

Java파일에 기능을 추가하고 다음과 같이 코드를 편집한다.

단계

1 firstServlet.java파일을 편집한다.

```
package firstservlet;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
public class firstservlet extends HttpServlet {
   private static final String CONTENT_TYPE = "text/html; charset=GB
K";
   //대역변수들을 초기화
   int connections;
   public void init(ServletConfig config)  throws ServletException {
   super.init(config);
   }
  //HTTP Get요청의 처리
   public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    //사용자이름
    String userName = request.getParameter("userName");
    if (userName == null) {
      userName = "hello,kim";
    }
    response.setContentType(CONTENT_TYPE);
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>firstservlet</title></head>");
    out.println("<body bgcolor=\"#ffffff\">");
    out.println("<p>The servlet has received a GET. This is the reply.
</p>");
    out.println("</body></html>");
  }
  //HTTP Post요청의 처리
  public void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    //사용자이름
```

```
        String userName = request.getParameter("userName");
        if (userName == null) {
          userName = "hello,kim";
        }
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head><title>firstservlet</title></head>");
        out.println("<body bgcolor=\"#ffffff\">");
        out.println("<p>The servlet has received a POST. This is the r
eply.</p>");
        out.print("<p>thanks for visiting!");
        out.println("</body></html>");
      }
      //HTTP Put요청의 처리
      public void doPut(HttpServletRequest request,
         HttpServletResponse response) throws ServletException,
         IOException {
        //사용자이름
        String userName = request.getParameter("param0");
        if (userName == null) {
          userName = "hello,kim";
        }
      }
      // HTTP Delete요청의 처리
      public void doDelete(HttpServletRequest request,
           HttpServletResponse response) throws ServletException,
           IOException {
        //사용자이름
        String userName = request.getParameter("userName");
        if (userName == null) {
          userName = "hello,kim";
        }
      }

      //자원지우기
```

```
    public void destroy() {
    }
  }
```

2 Make지령을 실행하여 응용프로그람을 콤파일한다.

7.4.1. Servlet시험

Servlet는 Tomcat봉사기에서 실행되며 웨브열람기에서 HTML페지를 입출력한다. Servlet시험순서는 아래와 같다.

단계

1 프로젝트판의 ServletServer.java에서 마우스오른쪽단추를 찰칵하고 [Run]지령을 찰칵한다. ServletServer를 실행하기 위하여서는 Tomcat봉사기를 기동한다.

2 Tomcat봉사기기동창문을 최소화한다.

3 다음 웨브열람기를 연다. 열람기에서 firstServlet.html파일을 실행시킨다.

4 현시된 창문에서 이름을 입력하고 【submit】단추를 찰칵한다.

Servlet가 실행되면 이름과 《HelloWorld》, 그리고 방문객번호가 함께 현시된다.

열람기의 주소창에는 《http:/localhost:8080/ firstServlet.firstServlet》을 입력하여야 한다.

5 프로그람이 정상으로 가동되였는가를 확인한 다음 Tomcat봉사기기동창문에서 닫기단추를 찰칵하면 프로그람실행을 완료한다.

JBuilder를 리용하여 작성한 Servlet는 여러가지 방법으로 시험해볼수 있다.

7.4.2. Servlet설치

Servlet를 웨브봉사기에 설치하려면 웨브봉사기에 대한 자료를 알아야 한다. 일단 응용프로그람을 설치했으면 같은 콤퓨터 혹은 같은 하위망에 있는 서로 다른 콤퓨터에서 의뢰기프로그람과 봉사기프로그람을 실행시킬수 있다.

7.4.3. 작성한 Servlet에 대한 리해

HttpServlet클라스는 서로 다른 종류의 HTTP련결을 정의해준다.

처음으로 Servlet가 호출될 때 HttpServlet.init(Servlet Configconf)가 실행된다. 이때 전체변수와 개별적인 망페지들의 요청에 의존하지 않는 자원들이 창조된다.

실례에서는 변수가 초기화되여 요청에 대한 추적을 진행하게 하였다.

HttpServlet.doPost(HttpServletRequest,HttpServletResponse)에서 두 파라메터는 Servlet와 봉사기프로그람사이에 정보를 교환하는 Java객체로서 Applet의 AppletContext객체와 류사하다.

또한 실례에서 HttpServletResponse는 하나의 ServletOutputStream을 얻어 이것으로 웨브열람기에 자료를 되돌려보내게 한다.

CGI에서 프로그람은 표준출구를 사용할수 있다.

ServletOutputStream에는 아래와 같은 메쏘드들이 있다.

- ServletOutputStream.println(String)
  열람기에 문자렬을 발송한다. 이때 문자렬은 행바꾸기기호를 포함할수 있다.
- ServletOutputStream.print(String)
  열람기에서 행렬바꾸기기호를 포함하지 않는 문자렬을 발송한다.
- ServletOutputStream.close()
  자료흐름을 닫는다.

제5절. Servlet와 JSP통신

Servlet는 Java를 리용하여 작성한 봉사기측프로그람으로서 가동기반에 의존하지 않는다. Servlet는 Java가동기반의 웨브봉사기에서 실행된다. Java Servlet는 Servlet의 능력을 동적으로 확장하며 요청-응답형식의 웨브봉사를 제공한다. JSP는 Java Server Pages의 략어로서 Sun회사가 출품한 웨브개발언어이다. JSP는 Microsoft회사의 ASP와 류사하다. 그러나 JSP는 가동기반에 의존하지 않는것으로하여 응용범위가 점점 넓어지고있다.

7.5.1. Servlet와 JSP결합의 개발방식

Java Server Pages(JSP)는 일반적인 정적페지과 동적페지를 결합하여 실현한 기술이다. CGI로 만들어진 많은 홈페지들은 그의 대부분이 정적페지이고 일부만이 동적페지이다. 그러나 Servlet를 포함하는 CGI기술은 원만한 동적페지를 작성할수 있다.

아래에서는 간단한 JSP페지를 주었다.

```
<!DOCTYPE HTML PUBLIC “-//W3C//DTD HTML4.0
 Transitional//EN”>
<HTML>
<HEAD><TITLE>우리 홈페지를 방문하신것을 환영합니다.
</TITLE></HEAD>
```

```
<BODY>
<HI>환영</HI>
<SMALL>환영,
<!...처음으로 방문하는 사용자이름은 “New User”...>
<% out.println(Utils.getUserNameFromCookie(reguest));%>
사용자정보를 설정하려면
<A HREF= “Account-Settings.html” >여기를 눌러주십시오.
</A></SMALL>
<P>
    홈페지의 다른 내용......,
</BODY></HTML>
```

JSP는 ASP에 비해 두가지 측면의 우점이 있다.

JSP는 우선 동적부분이 VB Script 혹은 Microsoft회사의 언어에 의해서가 아니라 Java로 작성된다. 다음 우점은 JSP는 여러 조작체계와 Microsoft가 아닌 다른 웨브봉사기에서 쓸수 있다는것이다.

JSP와 Servlet를 비교해보면 JSP에 Servlet로 실현할수 없는 기능이 추가된것은 없지만 HTML편집이 편리하다. JSP에서는 HTML의 매 행코드를 출구하는데 println을 리용하지 않아도 된다. 보다 중요한것은 망페지제작에서 기능별로 분리되여있는것이다. 즉 망페지전문가가 HTML을 설계하고 Servlet작성자가 동적내용부분을 설계할수 있다.

JSP와 JavaScript를 비교하여 볼 때 JavaScript는 의뢰기에서 동적으로 HTML을 작성할수 있는 언어로서 이것은 의뢰기환경을 기초로 하는 동적정보만을 처리할수 있다. 쿠키나 HTTP상태정보, 표가 제공하는 자료들은 JavaScript에서 사용할수 없다. 또한 의뢰기에서 실행하므로 자료기지, 등록부정보와 같은 봉사기자원에 접근할수 없다.

1. Servlet와 JSP개발도구의 설치

Servlet와 JSP개발을 하려면 먼저 JavaServlet 2.2/3.3과 JavaServerPages 1.2, Pages 1.2개발환경을 마련하여야 한다. Sun회사에서 무상으로 제공하는 Java Server Web Development Kit(JSWDK:Java봉사기웨브개발도구)는 http://java.sun.con/products/Servlet/에서 내리적재할수 있다.

JSWDK를 설치한 후에는 콤파일에 필요한 Servlet와 JSP클라스의 경로를 javac에서 지정해주어야 한다. 그러기 위하여 Servlet.jar와 jsp.jar를 CLASSPATH에 추가해주어야 한다. 또한 다른 개발자가 설치한 Servlet들과 충돌하지 않게 하려면 자기의 Servlet를 하나의 패키지로 묶어야 한다.

2. Servlet를 지원하는 웨브봉사기

개발도구를 준비한 다음에는 JavaServlet를 지원하는 웨브봉사기를 설치하거나 현재의 웨브봉사기에 Servlet패키지를 설치하여야 한다.

만일 현재 사용하고있는 웨브봉사기 혹은 응용프로그람봉사기가 최신판본이라면 많은 경우 필요한 프로그람들을 다 구축하고있을수 있다. 필요하다면 웨브봉사기의 도움말 혹은 http://java.sun.con/products/servlet/industry.html에서 Servlet봉사기와 관련한 구체적인 내용을 참고할수 있다.

아래에서는 현재 많이 리용되고있는 쏘프트웨어들을 소개한다.

- Apache Tomcat

Tomcat는 Servlet2.3과 JSP1.2를 실행하고 Servlet, JSP를 시험할수 있다. Tomcat 역시 Apache와 같은 무료쏘프트웨어이다. 그러나 Tomcat도 Apache와 같이 설치와 설정에서 불편한 약점을 가지고있다.

- Java Server Web Development Kit(JSWDK)

JSWDK로는 Servlet, JSP를 시험할수 있다. JSWDK는 무료쏘프트웨어이고 안정하지만 설치와 설정이 비교적 복잡하다. JSWDK에 대한 구체적인 내용은 http://java.sun.com/products/servlet/download.html에서 참고할수 있다.

이밖에도 AlIaire JRun, New Atlanta의 ServletExec, Gefion의 LiteWebServer(LWS) Sun의 Java Web Server 등이 있다.

7.5.2. Servlet와 JSP의 통신방법

1. 실례를 통한 Servlet의 기본구조에 대한 리해

Servlet는 GET와 POST요청을 쉽게 처리할수 있다. 실례에서는 GET요청을 Servlet로 처리하였다.

```
 import java.io.*;
 import javax.Servlet.*;
 import java.Servlet.http.*;
 public class SomeServlet extends HttpServlet{
 public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException{
        // “request”를 리용하여 요청과 관계되는 정보(실례로 쿠기)와 표값을
얻는다.
        // “response”를 리용하여 HTTP응답상태코드와 응답머리부를 지정
              printWriter out = response.getWriter();
        // “out”를 리용하여 응답내용을 열람기에 발송한다.
    }
   }
```

Servlet클라스는 반드시 HttpServlet를 계승하여야 하는데 자료가 GET 혹은 POST 방식으로 발송되는가에 따라 doGet 혹은 doPost메쏘드를 재정의하거나 이 두 메쏘드들을 다 재정의하여야 한다.

doGet와 doPost메쏘드에는 두개의 파라메터 즉 HttpServletRequest와 HttpServlet Response가 있다.

HttpServletRequest는 요청과 관련한 정보 례하면 표자료, HTTP요청머리부정보 등과 같은 요청정보를 얻는 메쏘드를 제공한다. HttpServletResponse는 HTTP응답상태(200, 404 등), 응답머리부(Const-Type, Set-Cookie 등)를 얻는 메쏘드를 제공해주며 중요하게는 의뢰기에 자료를 발송하는 PrintWriter를 제공해준다. 간단한 Servlet는 대부분 Println를 리용하여 정보를 의뢰기에 발송한다.

---

doGet와 doPost는 두가지 례외를 발생시키므로 선언부에서 처리해주어야 한다. 또한 java.io(PrintWriter 등 클라스를 포함)
javax.Servlet(HttpServlet 등 클라스를 포함)
javax.Servlet.http(HttpServletRequest HttpServlet Response클라스 포함)패키지를 인입해야 한다.

---

doGet와 doPost메쏘드들은 service메쏘드에 의해 호출되므로 Servlet가 GET와 POST요청을 처리해야 한다면 service메쏘드를 재정의하여야 한다.

2. 문자만 출력하는 간단한 Servlet의 작성

아래에서는 문자만 출력하는 간단한 Servlet작성실례를 주었다.

```
//HelloWorld.java
package hall;
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
public class HelloWorld extends HttpServlet{
   public void doGet(HttpServletRequest request,
       HttpServletResponse response)
       throws ServletException, IOException{
         PrintWriter out = response.getWriter();
         Out.println( “Hello World” );
     }
 }
```

3. Servlet의 콤파일과 설치

서로 다른 웨브봉사기들에 Servlet를 설치하는 방법은 다를수 있다. 이에 대해서는 구체적인 웨브봉사기의 도움말을 참고하면 된다. 례하면 만일 Java Web Server(JWS)2.0를 사용한다면 Servlet는 JWS설치등록부의 Servlets하위등록부에 설치하여야 한다.

우의 실례에서는 하나의 봉사기에서 서로 다른 사용자의 Servlet이름들이 충돌하는것을 막기 위하여 모든 Servlet를 하나의 독립적인 hall패키지에 넣었다. 만일 다른 사람과 하나의 봉사기를 함께 리용하고 이 봉사기에서 충돌을 피하기 위한 《가상봉사기》가 없다면 패키지를 리용하는것이 제일 합리적이다. Servlet를 hall패키지에 넣으면 HelloWorld.java는 Servlet목록의 hall하위등록부에 있는것과 같다.

JWS뿐아니라 대다수 다른 봉사기의 설정방법도 이와 류사하다.

Servlet클라스들을 콤파일할때 아래의 두가지 방법을 리용할수 있다.

첫째 방법은 CLASSPATH에서 Servlet가 보관되여있는 등록부(Servlet의 기본등록부)를 지정해주고 이 등록부에서 보통방식으로 콤파일하면 된다. 실례로 Servlet의 기본등록부가 《C:\JavaWebServerServlets》이고 패키지의 이름(기본등록부아래의 하위등록부이름)이 hall이라면 Windows에서 다음과 같이 콤파일한다.

```
DOS>Set CLASSPATH=C:\JavaWebServerServlets;%CLASSPATH%
DOS>cd C:\JavaWebServerServlets\hall
DOS>javac YourServlet.java
```

두번째 방법은 Servlet기본등록부에 들어간 다음 《 javac directory\YourServlet.java 》(Windows에서) 혹은 《javac directory/YourServlet.java》(Unix에서)지령으로 패키지안의 클라스를 콤파일한다. 실례로 Servlet기본등록부가 《C:\JavaWebServerServlets》이고 패키지이름이 hall이면 Windows에서 다음과 같이 콤파일한다.

```
DOS>cd C:\JavaWebServerServlets
DOS>Javac hall\YourServlet.java
```

콤파일을 진행할 때 javac의 《-d》파라메터를 리용하여 .class파일의 보관위치를 지정해줄수도 있다.

4. Servlet의 실행

Java Web Server에서는 Servlet를 JWS설치등록부의 Servlets하위등록부에 넣고 《http://host/Servlet/ServletName》으로 호출한다. 여기에서 주의해야 할것은 등록부이름은 Servlets로서 마지막에 《S》가 있지만 URL의 Servlet에는 《s》가 없다는것이다.

Hello World Servlet를 hall패키지에 넣었으므로 URL은 《 http://host/Servlet/hall.HelloWorld》로 되여야 한다.

기타 다른 봉사기에서는 Servlet를 설치하고 리용하는 방법이 조금씩 다르다. 대다수 웨

브봉사기는 Servlet의 별명을 정의한다. 때문에 Servlet의 URL이 《http://host/any-path/any-file.html/》형식으로 될수 있다.

5. HTML을 출력하는 Servlet

Servlet에서 HTML을 출력하려면 일반적으로 두가지 조작이 더 필요하다. 하나는 열람기에 발송하게 될 내용이 HTML이라는것을 알려주는것이며 다른 하나는 Println을 리용하여 합법적인 HTML페지를 만드는것이다.

먼저 HttpServletResponse의 setHeader메쏘드를 리용하여 Content-Type(내용류형)응답머리부를 설치하여야 한다. 그러나 Content-Type설치가 매우 복잡하므로 ServletAPI는 전용메쏘드 setContentType를 제공해준다.

주의: 응답머리부설치는 PrintWriter로 내용을 발송하기전에 진행하여야 한다.

례:

```
//HelloWWW.java
   package hall;
   import java.io.*;
   import javax.Servlet.*;
   import javax.Servlet.http.*;
   public class HelloWWW extends HttpServlet{
          public void doGet(HttpServletRequest request,
                HttpServletResponese response)
                  throws ServletException,IOException{
                       response.setcontentType( “text/html” );
                       PrintWriter out= response.getWriter();
                       out.println( “<!DOCTYPE HTML PUBLIC
                                      //W3C//DTD “HTML4.0 Transitional//E
                                      N>
                             “+” <HTML>
                             “+” <HEAD><TITLE>Hello WWW</TITLE></HEAD>
                             “+” <BODY>
                             “+” <H1>Hello WWW</H1>
                             “+” </BODY></HTML>” );
                    }
       }
```

6. 몇가지 HTML도구함수들

println를 리용하여 HTML를 출력하는것은 불편하므로 JSP를 리용한다. 일반적으로 Servlet에서 웨브페지의 DOCTYPE와 HEAD를 변화시킬수 없으므로 도구함수를 리용한다. 현재 리용되고있는 대다수의 열람기들은 DOCTYPE행을 무시하지만 HTML은 DOCTYPE행을 요구한다. 이것은 HTML문법검사기가 HTML의 합법성을 검사하는데 리용되기때문이다.

많은 웨브페지들에서 HEAD부분은 〈TITLE〉만을 포함하고있다. 그러므로 웨브페지작성자들은 HEAD에 많은 META표식과 양식선언을 리용하지만 여기서는 가장 간단한 경우만을 취급하려고 한다.

아래의 Java메쏘드는 망페지의 제목만을 파라메터로 하며 페지의 DOCTYPE , HEAD , TITLE부분을 출력한다.

```
//ServletUtilities.java
package hall;
public class ServletUtilities{
 public static final String DOCTYPE =
   “<!DOCTYPE HTML PUBLIC//W3C//DTD HTML4.0 Transitional //
EN>” ;
    public static String headWithTitle(String title){
          return(DOCTYPE+ “  ” + “<HTML>” +
             “<HEAD><TITLE>” +title+ “</TITLE></HEAD></HTML>” );
          }
         //다른 도구함수를 리용한 코드는 뒤에서 소개한다.
```

아래의 코드는 HelloWWW2클라스를 정의한것이다.

```
package hall;
import java.io.*;
import javax.Servlet.*;
import javax.Servlet.http.*;
public class HelloWWW2 extends HttpServlet{
       public void doGet(HttpServletRequest request,
              HttpServletResponse response)
              Throws ServletException,IOException{
                     response.setContentType( “text/html” );
                     printWriter out = response.getWriter();
                     out.println(ServletUtilities.headWithTitle
```

```
                    ( “HelloWWW” + “<HTML><BODY>” + “<H1>
                           HelloWWW</H1>” + “</BODY></HTML>” );
                    }
          }
```

제6절. Servlet와 3층 웨브구조

웨브기술이 발전함에 따라 전통적인 의뢰기/봉사기구조의 기업급응용체계는 점차적으로 열람기/중간층/자료기지봉사기의 3층구조로 바뀌여지고있다. 이러한 구조변화는 Microsoft의 Windows DNA에서도 Java가 핵심기술로 등장하게 하였다. Servlet기술의 출현은 Java를 핵심기술로 하는 기업급 3층웨브응용프로그람개발을 적극 추진시켰다. Servlet는 3층웨브봉사기에서 중간층을 실현하기에 가장 적합한 기술이다.

이 절에서는 기업급의 3층웨브구조를 구축하는데서 Servlet사용방법을 소개한다.

7.6.1. Servlet기술을 리용한 3층구조해결

기업급 웨브응용프로그람은 일반적으로 웨브열람기, 중간층(middle), 자료기지봉사기의 3개 층을 포함한다.

- 웨브열람기

웨브열람기는 3층구조의 첫번째 층으로서 의뢰기측의 열람기로 리용된다.

- Servlet중간층

중간층은 봉사기에서 실행되며 웨브열람기와 자료기지봉사기사이의 련계를 취해준다. 현재 중간층실현에서는 CGI, Java, Servlet 등을 리용한다. Servlet가 Java언어로 개발된것으로 하여 성능, 믿음성, 이식성 등에서 CGI보다 전망적이다. Servlet는 현재 중간층개발에서 가장 적합한 기술로 되고있다.

- 자료봉사기

자료봉사기는 사용자가 자료정보를 보관하는 곳으로서 중간층은 ODBC(CGI 중간층) 혹은 JDBC(Servlet중간층)를 리용하여 자료기지에 접근한다.

7.6.2. 3층구조로 된 웨브의 응용

아래에서는 간단한 3층 웨브응용실례로서 Servlet로 중간층을 실현하는 방법을 설명하였다. 실례에서는 간단한 도서관관리체계를 실현하고있다.

1. 사용자는 웨브열람기를 통하여 도서정보를 중간층에 넘겨주고 중간층이 이 정보를 자료기지에 저장한다.

2. 사용자는 웨브열람기를 통하여 도서정보를 검색하고 중간층에 의해 도서자료가 열람기에 현시된다.

체계는 3층구조의 웨브를 응용하고 자료기지봉사기는 Microsoft Access를 리용한다. 그리고 중간층은 하나의 Servlet이다. 이때 체계의 흐름공정은 다음과 같다.

단계

1 사용자는 HTML페지에 도서정보를 입력하고 입력된 자료는 중간층의 Book Servlet에 보내진다.

2 중간층 BookServlet는 SQL지령을 작성하며 SQL지령은 JDBC에 전달된다.

3 자료기지봉사기는 SQL지령을 집행하며 그 결과를 중간층 BookServlet에 돌려준다.

4 중간층 BookServlet는 자료기지에서 보내온 내용으로 HTML을 구축하여 의뢰기의 열람기에 보낸다.

전체체계에 대한 설계는 다음의 두 단계로 실현할수 있다.

- 자료기지구조 설계

복잡성을 피하기 위하여 자료기지는 BookTable표 하나만을 포함하며 표는 다음과 같다.

표 7-2. BookTable

이름	류형	길이	설명
bookname	String	50	도서명
isbn	String	50	도서번호

- 열람기의 HTML페지설계

사용자는 망페지에서 서고에 들어있는 도서를 찾거나 새로운 도서정보를 입력할수 있다.

3. 중간층 BookServlet설계

Servlet기술을 리용하여 실현하는 중간층은 열람기와 자료기지봉사기를 련결하는 역할을 한다. 이 중간층은 열람기의 HTML이 보내는 파라메터에 기초하여 필요한 SQL지령을 자료봉사기에 보낸다. 다음 SQL지령이 집행된 결과에 기초하여 HTML페지를 만들어 열람기에 전송한다.

7.6.3. 중간층의 실현

아래에 BookServlet를 실례로 중간층을 실현하는 방법에 대하여 보기로 한다.

1. Servlet초기화

코드는 아래와 같다.

```
  public class BookServlet extends HttpServlet{
   protected Connection dbConnection;
   protected PrepareStatement readQuery;
   protected PrepareStatement  writeQuery;
   protected String dbName = “jdbc:odbc:BookDatabase”;
   protected String bookName;
   protected String bookISBN;
   public void init(ServletConfig config) throws ServletException
   {
           try{
                Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
                   dbConnection=DriverManager.getConnection(dbNam
e,” ”,” ”);
           }
           catch(Exception e)
           {
                System.out.println(“Can not initialize database”);
           }
   }
```

Servlet가 처음 능동으로 될 때 Servlet의 init()메쏘드를 호출한다. Book Servlet는 init()에서 도서자료기지와의 련결을 창조한다.(물론 ODBC에서 bookDatabase가 정의되여야 한다.) 여기에서는 Java JDBC API의 Connection객체를 리용하였다.

2. Service()조작의 실현

의뢰기가 Servlet에 요청할 때 Servlet의 Service()메쏘드가 호출된다. 중간층의 모든 기능은 Service()에서 실현하여야 한다. 상응한 코드는 아래와 같다.

```
   Public void service(HttpServletRequest requese,HttpServletRespons
e response)
            Throws ServletException.IOException
{
   bookName=request.getPareameter(“BookName”);
   bookISBN=request.getParameter(“BookISBN”);
   if(bookName==null && bookISBN==null)
```

```
            doQueryBook(quest,response);
    else doNewBook(rquest,response);
}
```

코드에서는 먼저 파라메터 HttpServletReguest를 리용하여 의뢰기에서 입력한 자료를 얻는다. 이 자료들은 사용자가 HTML페지의 편집칸에 입력한 내용으로서 자료기지를 검색하거나 갱신하는 조작을 진행하는데 리용된다.

- 자료기지검색과 열람기에 결과를 보내기

자료기지검색은 우선 검색조건에 근거하여 SQL명령을 만들고 PrepareStatement객체를 설정한 다음 그것의 executeQuery()를 리용하여 자료기지에 자료조사를 의뢰한다. 결과를 얻은 다음 Servlet는 HTTPServletResponse를 리용하여 결과가 서술된 HTML페지를 만들어 열람기에 보낸다.

코드는 아래와 같다.

```
    Public void doQueryBook(HttpServletRequest request,HttpServletR
esponse response)
{
            try{
                    readQuery=dbConnection.preqareStatement( “SELEC
T FROM BOOKTABLE” ):
                    String htmlHead=” <html><head><title>검색결과</title>
<head>” ;
                    String htmlBody=” <body>” +;
                    resultSet readResult=readQuery.executeQuery();
                    while(readSet.next())
                    {
                    String Name=readResult.getString( “BookName” );
                      Strin ISBN=readResult.getString( “ISBN” );
                         HtmlBody+=Name+” ” +ISBN+” ” ;
                        }
                      htmlBody+=” </body>” +<html>”
                      PrintWriter output=new PrinterWrite(response.get
                              OutputStream());
                      Response.setContentType( “text/html” );
                      Output.println(htmlHead+htmlBody+htmlTail);
                        }
                      catch(Exception e)
```

```
                    {............}
          }
```

- 자료기지에 자료입력

자료기지에 자료를 입력하는 과정은 검색과 류사하게 SQL명령을 리용하여 만든다. 실례에서 SQL명령문은 다음과 같다.

String writeSql= “INSERT INTO BookTable(BookName,ISBN) Values(“+book Name+ “,” +book ISBN+ “,” ;

SQL명령을 실행한 다음에는 “Prepare Statement ::execute Update();”를 리용한다.

우에서 본 도서관관리실례와 같이 Servlet기술에 의한 중간층실현은 매우 편리하며 조작과정이 CGI를 리용할 때와 류사하다. 그러나 Servlet는 스레드에 기초하여 매 사용자들의 요청을 독립적인 과정으로 처리하지 않기때문에 CGI보다 우월하다. 또한 Servlet는 Java언어로 작성되므로 이식성, 코드재리용성도 CGI보다 높다. 그밖에 Servlet는 Sockets와 기타 다른 Servlet 혹은 Applet사이의 통신 등에 리용되여 복잡한 웨브응용을 실현할수 있다.

찾 아 보 기

JBuilder 배우기

집　필	강 철, 김영일	심　사	최광철
편　집	차현옥	교　정	서금석
장　정	서경애	콤퓨터편성	여은정
낸　곳	교육성 교육정보쎈터	인쇄소	교육성 교육정보쎈터
인　쇄	주체 97(2008)년 8월 10일	발　행	주체 97(2008)년 8월 20일

교-07-1311